Configuring Strimzi

Table of Contents

Chapter 1. Configuration overview

Strimzi simplifies the process of running Apache Kafka in a Kubernetes cluster.

This guide describes how to configure and manage a Strimzi deployment.

1.1. Configuring custom resources

Use custom resources to configure your Strimzi deployment.

You can use custom resources to configure and create instances of the following components:

- Kafka clusters
- Kafka Connect clusters
- Kafka MirrorMaker
- Kafka Bridge
- Cruise Control

You can also use custom resource configuration to manage your instances or modify your deployment to introduce additional features. This might include configuration that supports the following:

- Securing client access to Kafka brokers
- Accessing Kafka brokers from outside the cluster
- Creating topics
- Creating users (clients)
- Controlling feature gates
- Changing logging frequency
- Allocating resource limits and requests
- Introducing features, such as Strimzi Drain Cleaner, Cruise Control, or distributed tracing.

The Custom resource API reference describes the properties you can use in your configuration.

1.2. Using ConfigMaps to add configuration

Use `ConfigMap` resources to add specific configuration to your Strimzi deployment. ConfigMaps use key-value pairs to store non-confidential data. Configuration data added to ConfigMaps is maintained in one place and can be reused amongst components.

ConfigMaps can only store configuration data related to the following:

- Logging configuration
- Metrics configuration

- External configuration for Kafka Connect connectors

You can't use ConfigMaps for other areas of configuration.

When you configure a component, you can add a reference to a ConfigMap using the `configMapKeyRef` property.

For example, you can use `configMapKeyRef` to reference a ConfigMap that provides configuration for logging. You might use a ConfigMap to pass a Log4j configuration file. You add the reference to the `logging` configuration.

Example ConfigMap for logging

```
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: my-config-map
        key: my-config-map-key
```

To use a ConfigMap for metrics configuration, you add a reference to the `metricsConfig` configuration of the component in the same way.

`ExternalConfiguration` properties make data from a ConfigMap (or Secret) mounted to a pod available as environment variables or volumes. You can use external configuration data for the connectors used by Kafka Connect. The data might be related to an external data source, providing the values needed for the connector to communicate with that data source.

For example, you can use the `configMapKeyRef` property to pass configuration data from a ConfigMap as an environment variable.

Example ConfigMap providing environment variable values

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  externalConfiguration:
    env:
      - name: MY_ENVIRONMENT_VARIABLE
        valueFrom:
          configMapKeyRef:
            name: my-config-map
            key: my-key
```

If you are using ConfigMaps that are managed externally, use configuration providers to load the

data in the ConfigMaps.

1.2.1. Naming custom ConfigMaps

Strimzi creates its own ConfigMaps and other resources when it is deployed to Kubernetes. The ConfigMaps contain data necessary for running components. The ConfigMaps created by Strimzi must not be edited.

Make sure that any custom ConfigMaps you create do not have the same name as these default ConfigMaps. If they have the same name, they will be overwritten. For example, if your ConfigMap has the same name as the ConfigMap for the Kafka cluster, it will be overwritten when there is an update to the Kafka cluster.

Additional resources

- List of Kafka cluster resources (including ConfigMaps)
- Logging configuration
- `metricsConfig`
- `ExternalConfiguration` schema reference
- Loading configuration values from external sources

1.3. Document Conventions

User-replaced values

User-replaced values, also known as *replaceables*, are shown in *italics* with angle brackets (< >). Underscores (_) are used for multi-word values. If the value refers to code or commands, `monospace` is also used.

For example, in the following code, you will want to replace `<my_namespace>` with the name of your namespace:

```
sed -i 's/namespace: .*/namespace: <my_namespace>/' install/cluster-
operator/*RoleBinding*.yaml
```

1.4. Additional resources

- Strimzi Overview
- Deploying and Upgrading Strimzi
- Using the Strimzi Kafka Bridge

Chapter 2. Configuring a Strimzi deployment

Configure your Strimzi deployment using custom resources. Strimzi provides example configuration files, which can serve as a starting point when building your own Kafka component configuration for deployment.

NOTE	Labels applied to a custom resource are also applied to the Kubernetes resources making up its cluster. This provides a convenient mechanism for resources to be labeled as required.

Monitoring a Strimzi deployment

You can use Prometheus and Grafana to monitor your Strimzi deployment. For more information, see Introducing metrics to Kafka.

2.1. Kafka cluster configuration

Configure a Kafka deployment using the `Kafka` resource. A Kafka cluster is deployed with a ZooKeeper cluster, so configuration options are also available for ZooKeeper within the `Kafka` resource. The Entity Operator comprises the Topic Operator and User Operator. You can also configure `entityOperator` properties in the `Kafka` resource to include the Topic Operator and User Operator in the deployment.

`Kafka` schema reference describes the full schema of the `Kafka` resource.

For more information about Apache Kafka, see the Apache Kafka documentation.

Listener configuration

You configure listeners for connecting clients to Kafka brokers. For more information on configuring listeners, see `GenericKafkaListener` schema reference.

Managing TLS certificates

When deploying Kafka, the Cluster Operator automatically sets up and renews TLS certificates to enable encryption and authentication within your cluster. If required, you can manually renew the cluster and clients CA certificates before their renewal period starts. You can also replace the keys used by the cluster and clients CA certificates. For more information, see Renewing CA certificates manually and Replacing private keys.

2.1.1. Configuring Kafka

Use the properties of the `Kafka` resource to configure your Kafka deployment.

As well as configuring Kafka, you can add configuration for ZooKeeper and the Strimzi Operators. Common configuration properties, such as logging and healthchecks, are configured independently for each component.

This procedure shows only some of the possible configuration options, but those that are particularly important include:

- Resource requests (CPU / Memory)
- JVM options for maximum and minimum memory allocation
- Listeners (and authentication of clients)
- Authentication
- Storage
- Rack awareness
- Metrics
- Cruise Control for cluster rebalancing

Kafka versions

The `inter.broker.protocol.version` property for the Kafka `config` must be the version supported by the specified Kafka version (`spec.kafka.version`). The property represents the version of Kafka protocol used in a Kafka cluster.

From Kafka 3.0.0, when the `inter.broker.protocol.version` is set to `3.0` or higher, the `log.message.format.version` option is ignored and doesn't need to be set.

An update to the `inter.broker.protocol.version` is required when upgrading your Kafka version. For more information, see Upgrading Kafka.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

See the *Deploying and Upgrading Strimzi* guide for instructions on deploying a:

- Cluster Operator
- Kafka cluster

Procedure

1. Edit the `spec` properties for the `Kafka` resource.

   The properties you can configure are shown in this example configuration:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3 ①
    version: 3.4.0 ②
    logging: ③
      type: inline
      loggers:
        kafka.root.logger.level: "INFO"
```

```
    resources: ④
      requests:
        memory: 64Gi
        cpu: "8"
      limits:
        memory: 64Gi
        cpu: "12"
    readinessProbe: ⑤
      initialDelaySeconds: 15
      timeoutSeconds: 5
    livenessProbe:
      initialDelaySeconds: 15
      timeoutSeconds: 5
    jvmOptions: ⑥
      -Xms: 8192m
      -Xmx: 8192m
    image: my-org/my-image:latest ⑦
    listeners: ⑧
      - name: plain ⑨
        port: 9092 ⑩
        type: internal ⑪
        tls: false ⑫
        configuration:
          useServiceDnsDomain: true ⑬
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication: ⑭
          type: tls
      - name: external ⑮
        port: 9094
        type: route
        tls: true
        configuration:
          brokerCertChainAndKey: ⑯
            secretName: my-secret
            certificate: my-certificate.crt
            key: my-key.key
    authorization: ⑰
      type: simple
    config: ⑱
      auto.create.topics.enable: "false"
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      default.replication.factor: 3
      min.insync.replicas: 2
      inter.broker.protocol.version: "3.4"
    storage: ⑲
      type: persistent-claim ⑳
```

```
      size: 10000Gi
    rack:
      topologyKey: topology.kubernetes.io/zone
    metricsConfig:
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: my-config-map
          key: my-key
    # ...
  zookeeper:
    replicas: 3
    logging:
      type: inline
      loggers:
        zookeeper.root.logger: "INFO"
    resources:
      requests:
        memory: 8Gi
        cpu: "2"
      limits:
        memory: 8Gi
        cpu: "2"
    jvmOptions:
      -Xms: 4096m
      -Xmx: 4096m
    storage:
      type: persistent-claim
      size: 1000Gi
    metricsConfig:
      # ...
  entityOperator:
    tlsSidecar:
      resources:
        requests:
          cpu: 200m
          memory: 64Mi
        limits:
          cpu: 500m
          memory: 128Mi
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: "INFO"
      resources:
        requests:
          memory: 512Mi
          cpu: "1"
```

```
        limits:
          memory: 512Mi
          cpu: "1"
    userOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
      resources:
        requests:
          memory: 512Mi
          cpu: "1"
        limits:
          memory: 512Mi
          cpu: "1"
  kafkaExporter:
    # ...
  cruiseControl:
    # ...
```

① The number of replica nodes.

② Kafka version, which can be changed to a supported version by following the upgrade procedure.

③ Kafka loggers and log levels added directly (`inline`) or indirectly (`external`) through a ConfigMap. A custom ConfigMap must be placed under the `log4j.properties` key. For the Kafka `kafka.root.logger.level` logger, you can set the log level to INFO, ERROR, WARN, TRACE, DEBUG, FATAL or OFF.

④ Requests for reservation of supported resources, currently `cpu` and `memory`, and limits to specify the maximum resources that can be consumed.

⑤ Healthchecks to know when to restart a container (liveness) and when a container can accept traffic (readiness).

⑥ JVM configuration options to optimize performance for the Virtual Machine (VM) running Kafka.

⑦ ADVANCED OPTION: Container image configuration, which is recommended only in special situations.

⑧ Listeners configure how clients connect to the Kafka cluster via bootstrap addresses. Listeners are configured as *internal* or *external* listeners for connection from inside or outside the Kubernetes cluster.

⑨ Name to identify the listener. Must be unique within the Kafka cluster.

⑩ Port number used by the listener inside Kafka. The port number has to be unique within a given Kafka cluster. Allowed port numbers are 9092 and higher with the exception of ports 9404 and 9999, which are already used for Prometheus and JMX. Depending on the listener type, the port number might not be the same as the port number that connects Kafka clients.

⑪ Listener type specified as `internal` or `cluster-ip` (to expose Kafka using per-broker `ClusterIP` services), or for external listeners, as `route` (OpenShift only), `loadbalancer`, `nodeport` or `ingress` (Kubernetes only).

⑫ Enables TLS encryption for each listener. Default is `false`. TLS encryption is not required for `route` listeners.

⑬ Defines whether the fully-qualified DNS names including the cluster service suffix (usually `.cluster.local`) are assigned.

⑭ Listener authentication mechanism specified as mTLS, SCRAM-SHA-512, or token-based OAuth 2.0.

⑮ External listener configuration specifies how the Kafka cluster is exposed outside Kubernetes, such as through a `route`, `loadbalancer` or `nodeport`.

⑯ Optional configuration for a Kafka listener certificate managed by an external CA (certificate authority). The `brokerCertChainAndKey` specifies a `Secret` that contains a server certificate and a private key. You can configure Kafka listener certificates on any listener with enabled TLS encryption.

⑰ Authorization enables simple, OAUTH 2.0, or OPA authorization on the Kafka broker. Simple authorization uses the `AclAuthorizer` Kafka plugin.

⑱ Broker configuration. Standard Apache Kafka configuration may be provided, restricted to those properties not managed directly by Strimzi.

⑲ Storage size for persistent volumes may be increased and additional volumes may be added to JBOD storage.

⑳ Persistent storage has additional configuration options, such as a storage `id` and `class` for dynamic volume provisioning.

Rack awareness configuration to spread replicas across different racks, data centers, or availability zones. The `topologyKey` must match a node label containing the rack ID. The example used in this configuration specifies a zone using the standard `topology.kubernetes.io/zone` label.

Prometheus metrics enabled. In this example, metrics are configured for the Prometheus JMX Exporter (the default metrics exporter).

Prometheus rules for exporting metrics to a Grafana dashboard through the Prometheus JMX Exporter, which are enabled by referencing a ConfigMap containing configuration for the Prometheus JMX exporter. You can enable metrics without further configuration using a reference to a ConfigMap containing an empty file under `metricsConfig.valueFrom.configMapKeyRef.key`.

ZooKeeper-specific configuration, which contains properties similar to the Kafka configuration.

The number of ZooKeeper nodes. ZooKeeper clusters or ensembles usually run with an odd number of nodes, typically three, five, or seven. The majority of nodes must be available in order to maintain an effective quorum. If the ZooKeeper cluster loses its quorum, it will stop responding to clients and the Kafka brokers will stop working. Having a stable and highly available ZooKeeper cluster is crucial for Strimzi.

Specified ZooKeeper loggers and log levels.

Entity Operator configuration, which specifies the configuration for the Topic Operator and User Operator.

Entity Operator TLS sidecar configuration. Entity Operator uses the TLS sidecar for secure communication with ZooKeeper.

Specified Topic Operator loggers and log levels. This example uses `inline` logging.

Specified User Operator loggers and log levels.

Kafka Exporter configuration. Kafka Exporter is an optional component for extracting metrics data from Kafka brokers, in particular consumer lag data. For Kafka Exporter to be able to work properly, consumer groups need to be in use.

Optional configuration for Cruise Control, which is used to rebalance the Kafka cluster.

2. Create or update the resource:

```
kubectl apply -f <kafka_configuration_file>
```

Default ZooKeeper configuration values

When deploying ZooKeeper with Strimzi, some of the default configuration set by Strimzi differs from the standard ZooKeeper defaults. This is because Strimzi sets a number of ZooKeeper properties with values that are optimized for running ZooKeeper within a Kubernetes environment.

The default configuration for key ZooKeeper properties in Strimzi is as follows:

Table 1. Default ZooKeeper Properties in Strimzi

Property	Default value	Description
`tickTime`	2000	The length of a single tick in milliseconds, which determines the length of a session timeout.
`initLimit`	5	The maximum number of ticks that a follower is allowed to fall behind the leader in a ZooKeeper cluster.
`syncLimit`	2	The maximum number of ticks that a follower is allowed to be out of sync with the leader in a ZooKeeper cluster.
`autopurge.purgeInterval`	1	Enables the `autopurge` feature and sets the time interval in hours for purging the server-side ZooKeeper transaction log.
`admin.enableServer`	false	Flag to disable the ZooKeeper admin server. The admin server is not used by Strimzi.

IMPORTANT Modifying these default values as `zookeeper.config` in the `Kafka` custom resource may impact the behavior and performance of your ZooKeeper cluster.

2.1.2. Configuring the Entity Operator

The Entity Operator is responsible for managing Kafka-related entities in a running Kafka cluster.

The Entity Operator comprises the:

- Topic Operator to manage Kafka topics
- User Operator to manage Kafka users

Through `Kafka` resource configuration, the Cluster Operator can deploy the Entity Operator, including one or both operators, when deploying a Kafka cluster.

The operators are automatically configured to manage the topics and users of the Kafka cluster. The Topic Operator and User Operator can only watch a single namespace.

NOTE	When deployed, the Entity Operator pod contains the operators according to the deployment configuration.

Entity Operator configuration properties

Use the `entityOperator` property in `Kafka.spec` to configure the Entity Operator.

The `entityOperator` property supports several sub-properties:

- `tlsSidecar`
- `topicOperator`
- `userOperator`
- `template`

The `tlsSidecar` property contains the configuration of the TLS sidecar container, which is used to communicate with ZooKeeper.

The `template` property contains the configuration of the Entity Operator pod, such as labels, annotations, affinity, and tolerations. For more information on configuring templates, see Customizing Kubernetes resources.

The `topicOperator` property contains the configuration of the Topic Operator. When this option is missing, the Entity Operator is deployed without the Topic Operator.

The `userOperator` property contains the configuration of the User Operator. When this option is missing, the Entity Operator is deployed without the User Operator.

For more information on the properties used to configure the Entity Operator, see the `EntityUserOperatorSpec` schema reference.

Example of basic configuration enabling both operators

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
```

```
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

If an empty object (`{}`) is used for the `topicOperator` and `userOperator`, all properties use their default values.

When both `topicOperator` and `userOperator` properties are missing, the Entity Operator is not deployed.

Topic Operator configuration properties

Topic Operator deployment can be configured using additional options inside the `topicOperator` object. The following properties are supported:

`watchedNamespace`

The Kubernetes namespace in which the Topic Operator watches for `KafkaTopic` resources. Default is the namespace where the Kafka cluster is deployed.

`reconciliationIntervalSeconds`

The interval between periodic reconciliations in seconds. Default `120`.

`zookeeperSessionTimeoutSeconds`

The ZooKeeper session timeout in seconds. Default `18`.

`topicMetadataMaxAttempts`

The number of attempts at getting topic metadata from Kafka. The time between each attempt is defined as an exponential back-off. Consider increasing this value when topic creation might take more time due to the number of partitions or replicas. Default `6`.

`image`

The `image` property can be used to configure the container image which will be used. For more details about configuring custom container images, see `image`.

`resources`

The `resources` property configures the amount of resources allocated to the Topic Operator. For more details about resource request and limit configuration, see `resources`.

`logging`

The `logging` property configures the logging of the Topic Operator. For more details, see `logging`.

Example Topic Operator configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
    # ...
```

User Operator configuration properties

User Operator deployment can be configured using additional options inside the `userOperator` object. The following properties are supported:

`watchedNamespace`

The Kubernetes namespace in which the User Operator watches for `KafkaUser` resources. Default is the namespace where the Kafka cluster is deployed.

`reconciliationIntervalSeconds`

The interval between periodic reconciliations in seconds. Default `120`.

`image`

The `image` property can be used to configure the container image which will be used. For more details about configuring custom container images, see `image`.

`resources`

The `resources` property configures the amount of resources allocated to the User Operator. For more details about resource request and limit configuration, see `resources`.

`logging`

The `logging` property configures the logging of the User Operator. For more details, see `logging`.

`secretPrefix`

The `secretPrefix` property adds a prefix to the name of all Secrets created from the KafkaUser resource. For example, `secretPrefix: kafka-` would prefix all Secret names with `kafka-`. So a KafkaUser named `my-user` would create a Secret named `kafka-my-user`.

Example User Operator configuration

```
apiVersion: kafka.strimzi.io/v1beta2
```

```
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      watchedNamespace: my-user-namespace
      reconciliationIntervalSeconds: 60
    # ...
```

2.1.3. Configuring Kafka and ZooKeeper storage

As stateful applications, Kafka and ZooKeeper store data on disk. Strimzi supports three storage types for this data:

- Ephemeral (Recommended for development only)
- Persistent
- JBOD (**Kafka only** not ZooKeeper)

When configuring a `Kafka` resource, you can specify the type of storage used by the Kafka broker and its corresponding ZooKeeper node. You configure the storage type using the `storage` property in the following resources:

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`

The storage type is configured in the `type` field.

Refer to the schema reference for more information on storage configuration properties:

- `EphemeralStorage` schema reference
- `PersistentClaimStorage` schema reference
- `JbodStorage` schema reference

WARNING	The storage type cannot be changed after a Kafka cluster is deployed.

Data storage considerations

For Strimzi to work well, an efficient data storage infrastructure is essential. We strongly recommend using block storage. Strimzi is only tested for use with block storage. File storage, such as NFS, is not tested and there is no guarantee it will work.

Choose one of the following options for your block storage:

- A cloud-based block storage solution, such as Amazon Elastic Block Store (EBS)
- Persistent storage using local persistent volumes
- Storage Area Network (SAN) volumes accessed by a protocol such as *Fibre Channel* or *iSCSI*

NOTE	Strimzi does not require Kubernetes raw block volumes.

File systems

Kafka uses a file system for storing messages. Strimzi is compatible with the XFS and ext4 file systems, which are commonly used with Kafka. Consider the underlying architecture and requirements of your deployment when choosing and setting up your file system.

For more information, refer to Filesystem Selection in the Kafka documentation.

Disk usage

Use separate disks for Apache Kafka and ZooKeeper.

Solid-state drives (SSDs), though not essential, can improve the performance of Kafka in large clusters where data is sent to and received from multiple topics asynchronously. SSDs are particularly effective with ZooKeeper, which requires fast, low latency data access.

NOTE	You do not need to provision replicated storage because Kafka and ZooKeeper both have built-in data replication.

Ephemeral storage

Ephemeral data storage is transient. All pods on a node share a local ephemeral storage space. Data is retained for as long as the pod that uses it is running. The data is lost when a pod is deleted. Although a pod can recover data in a highly available environment.

Because of its transient nature, ephemeral storage is only recommended for development and testing.

Ephemeral storage uses `emptyDir` volumes to store data. An `emptyDir` volume is created when a pod is assigned to a node. You can set the total amount of storage for the `emptyDir` using the `sizeLimit` property .

IMPORTANT	Ephemeral storage is not suitable for single-node ZooKeeper clusters or Kafka topics with a replication factor of 1.

To use ephemeral storage, you set the storage type configuration in the `Kafka` or `ZooKeeper` resource to `ephemeral`.

Example ephemeral storage configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
```

```
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: ephemeral
    # ...
  zookeeper:
    # ...
    storage:
      type: ephemeral
    # ...
```

Mount path of Kafka log directories

The ephemeral volume is used by Kafka brokers as log directories mounted into the following path:

```
/var/lib/kafka/data/kafka-logIDX
```

Where `IDX` is the Kafka broker pod index. For example `/var/lib/kafka/data/kafka-log0`.

Persistent storage

Persistent data storage retains data in the event of system disruption. For pods that use persistent data storage, data is persisted across pod failures and restarts.

A dynamic provisioning framework enables clusters to be created with persistent storage. Pod configuration uses Persistent Volume Claims (PVCs) to make storage requests on persistent volumes (PVs). PVs are storage resources that represent a storage volume. PVs are independent of the pods that use them. The PVC requests the amount of storage required when a pod is being created. The underlying storage infrastructure of the PV does not need to be understood. If a PV matches the storage criteria, the PVC is bound to the PV.

Because of its permanent nature, persistent storage is recommended for production.

PVCs can request different types of persistent storage by specifying a StorageClass. Storage classes define storage profiles and dynamically provision PVs. If a storage class is not specified, the default storage class is used. Persistent storage options might include SAN storage types or local persistent volumes.

To use persistent storage, you set the storage type configuration in the `Kafka` or `ZooKeeper` resource to `persistent-claim`.

In the production environment, the following configuration is recommended:

- For Kafka, configure `type: jbod` with one or more `type: persistent-claim` volumes
- For ZooKeeper, configure `type: persistent-claim`

Persistent storage also has the following configuration options:

`id` **(optional)**

A storage identification number. This option is mandatory for storage volumes defined in a JBOD storage declaration. Default is `0`.

`size` **(required)**

The size of the persistent volume claim, for example, "1000Gi".

`class` **(optional)**

The Kubernetes StorageClass to use for dynamic volume provisioning. Storage `class` configuration includes parameters that describe the profile of a volume in detail.

`selector` **(optional)**

Configuration to specify a specific PV. Provides key:value pairs representing the labels of the volume selected.

`deleteClaim` **(optional)**

Boolean value to specify whether the PVC is deleted when the cluster is uninstalled. Default is `false`.

WARNING	Increasing the size of persistent volumes in an existing Strimzi cluster is only supported in Kubernetes versions that support persistent volume resizing. The persistent volume to be resized must use a storage class that supports volume expansion. For other versions of Kubernetes and storage classes that do not support volume expansion, you must decide the necessary storage size before deploying the cluster. Decreasing the size of existing persistent volumes is not possible.

Example persistent storage configuration for Kafka and ZooKeeper

```
# ...
spec:
  kafka:
    # ...
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
      - id: 1
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
      - id: 2
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
    # ...
```

```
  zookeeper:
    storage:
      type: persistent-claim
      size: 1000Gi
# ...
```

If you do not specify a storage class, the default is used. The following example specifies a storage class.

Example persistent storage configuration with specific storage class

```
# ...
storage:
  type: persistent-claim
  size: 1Gi
  class: my-storage-class
# ...
```

Use a `selector` to specify a labeled persistent volume that provides certain features, such as an SSD.

Example persistent storage configuration with selector

```
# ...
storage:
  type: persistent-claim
  size: 1Gi
  selector:
    hdd-type: ssd
  deleteClaim: true
# ...
```

Storage class overrides

Instead of using the default storage class, you can specify a different storage class for one or more Kafka brokers or ZooKeeper nodes. This is useful, for example, when storage classes are restricted to different availability zones or data centers. You can use the `overrides` field for this purpose.

In this example, the default storage class is named `my-storage-class`:

Example Strimzi cluster using storage class overrides

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  labels:
    app: my-cluster
  name: my-cluster
  namespace: myproject
spec:
  # ...
```

```
kafka:
  replicas: 3
  storage:
    type: jbod
    volumes:
    - id: 0
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
      class: my-storage-class
      overrides:
      - broker: 0
        class: my-storage-class-zone-1a
      - broker: 1
        class: my-storage-class-zone-1b
      - broker: 2
        class: my-storage-class-zone-1c
    # ...
# ...
zookeeper:
  replicas: 3
  storage:
    deleteClaim: true
    size: 100Gi
    type: persistent-claim
    class: my-storage-class
    overrides:
      - broker: 0
        class: my-storage-class-zone-1a
      - broker: 1
        class: my-storage-class-zone-1b
      - broker: 2
        class: my-storage-class-zone-1c
# ...
```

As a result of the configured `overrides` property, the volumes use the following storage classes:

- The persistent volumes of ZooKeeper node 0 use `my-storage-class-zone-1a`.
- The persistent volumes of ZooKeeper node 1 use `my-storage-class-zone-1b`.
- The persistent volumes of ZooKeeepr node 2 use `my-storage-class-zone-1c`.
- The persistent volumes of Kafka broker 0 use `my-storage-class-zone-1a`.
- The persistent volumes of Kafka broker 1 use `my-storage-class-zone-1b`.
- The persistent volumes of Kafka broker 2 use `my-storage-class-zone-1c`.

The `overrides` property is currently used only to override storage class configurations. Overrides for other storage configuration properties is not currently supported. Other storage configuration properties are currently not supported.

PVC resources for persistent storage

When persistent storage is used, it creates PVCs with the following names:

`data-cluster-name-kafka-idx`
: PVC for the volume used for storing data for the Kafka broker pod `idx`.

`data-cluster-name-zookeeper-idx`
: PVC for the volume used for storing data for the ZooKeeper node pod `idx`.

Mount path of Kafka log directories

The persistent volume is used by the Kafka brokers as log directories mounted into the following path:

```
/var/lib/kafka/data/kafka-logIDX
```

Where `IDX` is the Kafka broker pod index. For example `/var/lib/kafka/data/kafka-log0`.

Resizing persistent volumes

You can provision increased storage capacity by increasing the size of the persistent volumes used by an existing Strimzi cluster. Resizing persistent volumes is supported in clusters that use either a single persistent volume or multiple persistent volumes in a JBOD storage configuration.

NOTE	You can increase but not decrease the size of persistent volumes. Decreasing the size of persistent volumes is not currently supported in Kubernetes.

Prerequisites

- A Kubernetes cluster with support for volume resizing.
- The Cluster Operator is running.
- A Kafka cluster using persistent volumes created using a storage class that supports volume expansion.

Procedure

1. Edit the `Kafka` resource for your cluster.

   Change the `size` property to increase the size of the persistent volume allocated to a Kafka cluster, a ZooKeeper cluster, or both.

   - For Kafka clusters, update the `size` property under `spec.kafka.storage`.
   - For ZooKeeper clusters, update the `size` property under `spec.zookeeper.storage`.

   Kafka configuration to increase the volume size to `2000Gi`

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: Kafka
   metadata:
   ```

```
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: persistent-claim
      size: 2000Gi
      class: my-storage-class
    # ...
  zookeeper:
    # ...
```

2. Create or update the resource:

```
kubectl apply -f <kafka_configuration_file>
```

Kubernetes increases the capacity of the selected persistent volumes in response to a request from the Cluster Operator. When the resizing is complete, the Cluster Operator restarts all pods that use the resized persistent volumes. This happens automatically.

3. Verify that the storage capacity has increased for the relevant pods on the cluster:

```
kubectl get pv
```

Kafka broker pods with increased storage

```
NAME               CAPACITY   CLAIM
pvc-0ca459ce-...   2000Gi     my-project/data-my-cluster-kafka-2
pvc-6e1810be-...   2000Gi     my-project/data-my-cluster-kafka-0
pvc-82dc78c9-...   2000Gi     my-project/data-my-cluster-kafka-1
```

The output shows the names of each PVC associated with a broker pod.

Additional resources

- For more information about resizing persistent volumes in Kubernetes, see Resizing Persistent Volumes using Kubernetes.

JBOD storage

You can configure Strimzi to use JBOD, a data storage configuration of multiple disks or volumes. JBOD is one approach to providing increased data storage for Kafka brokers. It can also improve performance.

NOTE	JBOD storage is supported for **Kafka only** not ZooKeeper.

A JBOD configuration is described by one or more volumes, each of which can be either ephemeral or persistent. The rules and constraints for JBOD volume declarations are the same as those for

ephemeral and persistent storage. For example, you cannot decrease the size of a persistent storage volume after it has been provisioned, or you cannot change the value of `sizeLimit` when the type is `ephemeral`.

To use JBOD storage, you set the storage type configuration in the `Kafka` resource to `jbod`. The `volumes` property allows you to describe the disks that make up your JBOD storage array or configuration.

Example JBOD storage configuration

```
# ...
storage:
  type: jbod
  volumes:
  - id: 0
    type: persistent-claim
    size: 100Gi
    deleteClaim: false
  - id: 1
    type: persistent-claim
    size: 100Gi
    deleteClaim: false
# ...
```

The IDs cannot be changed once the JBOD volumes are created. You can add or remove volumes from the JBOD configuration.

PVC resource for JBOD storage

When persistent storage is used to declare JBOD volumes, it creates a PVC with the following name:

`data-id-cluster-name-kafka-idx`

PVC for the volume used for storing data for the Kafka broker pod `idx`. The `id` is the ID of the volume used for storing data for Kafka broker pod.

Mount path of Kafka log directories

The JBOD volumes are used by Kafka brokers as log directories mounted into the following path:

```
/var/lib/kafka/data-id/kafka-logidx
```

Where `id` is the ID of the volume used for storing data for Kafka broker pod `idx`. For example `/var/lib/kafka/data-0/kafka-log0`.

Adding volumes to JBOD storage

This procedure describes how to add volumes to a Kafka cluster configured to use JBOD storage. It cannot be applied to Kafka clusters configured to use any other storage type.

NOTE	When adding a new volume under an `id` which was already used in the past and removed, you have to make sure that the previously used `PersistentVolumeClaims` have been deleted.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator
- A Kafka cluster with JBOD storage

Procedure

1. Edit the `spec.kafka.storage.volumes` property in the `Kafka` resource. Add the new volumes to the `volumes` array. For example, add the new volume with id `2`:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
      - id: 1
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
      - id: 2
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
    # ...
  zookeeper:
    # ...
```

2. Create or update the resource:

```
kubectl apply -f <kafka_configuration_file>
```

3. Create new topics or reassign existing partitions to the new disks.

TIP	Cruise Control is an effective tool for reassigning partitions. To perform an intra-

broker disk balance, you set `rebalanceDisk` to `true` under the `KafkaRebalance.spec`.

Removing volumes from JBOD storage

This procedure describes how to remove volumes from Kafka cluster configured to use JBOD storage. It cannot be applied to Kafka clusters configured to use any other storage type. The JBOD storage always has to contain at least one volume.

IMPORTANT	To avoid data loss, you have to move all partitions before removing the volumes.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator
- A Kafka cluster with JBOD storage with two or more volumes

Procedure

1. Reassign all partitions from the disks which are you going to remove. Any data in partitions still assigned to the disks which are going to be removed might be lost.

TIP	You can use the `kafka-reassign-partitions.sh` tool to reassign the partitions.

2. Edit the `spec.kafka.storage.volumes` property in the `Kafka` resource. Remove one or more volumes from the `volumes` array. For example, remove the volumes with ids `1` and `2`:

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: Kafka
   metadata:
     name: my-cluster
   spec:
     kafka:
       # ...
       storage:
         type: jbod
         volumes:
         - id: 0
           type: persistent-claim
           size: 100Gi
           deleteClaim: false
       # ...
     zookeeper:
       # ...
   ```

3. Create or update the resource:

   ```
   kubectl apply -f <kafka_configuration_file>
   ```

2.1.4. Retrieving JMX metrics with JmxTrans

As of Strimzi 0.35.0, support for JmxTrans has been removed.

2.1.5. Connecting to ZooKeeper from a terminal

Most Kafka CLI tools can connect directly to Kafka, so under normal circumstances you should not need to connect to ZooKeeper. ZooKeeper services are secured with encryption and authentication and are not intended to be used by external applications that are not part of Strimzi.

However, if you want to use Kafka CLI tools that require a connection to ZooKeeper, you can use a terminal inside a ZooKeeper container and connect to `localhost:12181` as the ZooKeeper address.

Prerequisites

- A Kubernetes cluster is available.
- A Kafka cluster is running.
- The Cluster Operator is running.

Procedure

1. Open the terminal using the Kubernetes console or run the `exec` command from your CLI.

   For example:

   ```
   kubectl exec -ti my-cluster-zookeeper-0 -- bin/kafka-topics.sh --list --zookeeper
   localhost:12181
   ```

   Be sure to use `localhost:12181`.

   You can now run Kafka commands to ZooKeeper.

2.1.6. Deleting Kafka nodes manually

This procedure describes how to delete an existing Kafka node by using a Kubernetes annotation. Deleting a Kafka node consists of deleting both the `Pod` on which the Kafka broker is running and the related `PersistentVolumeClaim` (if the cluster was deployed with persistent storage). After deletion, the `Pod` and its related `PersistentVolumeClaim` are recreated automatically.

WARNING	Deleting a `PersistentVolumeClaim` can cause permanent data loss and the availability of your cluster cannot be guaranteed. The following procedure should only be performed if you have encountered storage issues.

Prerequisites

See the *Deploying and Upgrading Strimzi* guide for instructions on running a:

- Cluster Operator
- Kafka cluster

Procedure

1. Find the name of the `Pod` that you want to delete.

   Kafka broker pods are named <*cluster-name*>-kafka-<*index*>, where <*index*> starts at zero and ends at the total number of replicas minus one. For example, `my-cluster-kafka-0`.

2. Annotate the `Pod` resource in Kubernetes.

   Use `kubectl annotate`:

   ```
   kubectl annotate pod cluster-name-kafka-index strimzi.io/delete-pod-and-pvc=true
   ```

3. Wait for the next reconciliation, when the annotated pod with the underlying persistent volume claim will be deleted and then recreated.

2.1.7. Deleting ZooKeeper nodes manually

This procedure describes how to delete an existing ZooKeeper node by using a Kubernetes annotation. Deleting a ZooKeeper node consists of deleting both the `Pod` on which ZooKeeper is running and the related `PersistentVolumeClaim` (if the cluster was deployed with persistent storage). After deletion, the `Pod` and its related `PersistentVolumeClaim` are recreated automatically.

WARNING	Deleting a `PersistentVolumeClaim` can cause permanent data loss and the availability of your cluster cannot be guaranteed. The following procedure should only be performed if you have encountered storage issues.

Prerequisites

See the *Deploying and Upgrading Strimzi* guide for instructions on running a:

- Cluster Operator
- Kafka cluster

Procedure

1. Find the name of the `Pod` that you want to delete.

   ZooKeeper pods are named <*cluster-name*>-zookeeper-<*index*>, where <*index*> starts at zero and ends at the total number of replicas minus one. For example, `my-cluster-zookeeper-0`.

2. Annotate the `Pod` resource in Kubernetes.

   Use `kubectl annotate`:

   ```
   kubectl annotate pod cluster-name-zookeeper-index strimzi.io/delete-pod-and-
   pvc=true
   ```

3. Wait for the next reconciliation, when the annotated pod with the underlying persistent volume claim will be deleted and then recreated.

2.1.8. List of Kafka cluster resources

The following resources are created by the Cluster Operator in the Kubernetes cluster:

Shared resources

`cluster-name-cluster-ca`

Secret with the Cluster CA private key used to encrypt the cluster communication.

`cluster-name-cluster-ca-cert`

Secret with the Cluster CA public key. This key can be used to verify the identity of the Kafka brokers.

`cluster-name-clients-ca`

Secret with the Clients CA private key used to sign user certificates

`cluster-name-clients-ca-cert`

Secret with the Clients CA public key. This key can be used to verify the identity of the Kafka users.

`cluster-name-cluster-operator-certs`

Secret with Cluster operators keys for communication with Kafka and ZooKeeper.

ZooKeeper nodes

`cluster-name-zookeeper`

Name given to the following ZooKeeper resources:

- StrimziPodSet for managing the ZooKeeper node pods.
- Service account used by the ZooKeeper nodes.
- PodDisruptionBudget configured for the ZooKeeper nodes.

`cluster-name-zookeeper-idx`

Pods created by the StrimziPodSet.

`cluster-name-zookeeper-nodes`

Headless Service needed to have DNS resolve the ZooKeeper pods IP addresses directly.

`cluster-name-zookeeper-client`

Service used by Kafka brokers to connect to ZooKeeper nodes as clients.

`cluster-name-zookeeper-config`

ConfigMap that contains the ZooKeeper ancillary configuration, and is mounted as a volume by the ZooKeeper node pods.

`cluster-name-zookeeper-nodes`

Secret with ZooKeeper node keys.

`cluster-name-network-policy-zookeeper`

Network policy managing access to the ZooKeeper services.

data-cluster-name-zookeeper-idx

Persistent Volume Claim for the volume used for storing data for the ZooKeeper node pod `idx`. This resource will be created only if persistent storage is selected for provisioning persistent volumes to store data.

Kafka brokers

cluster-name-kafka

Name given to the following Kafka resources:

- StrimziPodSet for managing the Kafka broker pods.
- Service account used by the Kafka pods.
- PodDisruptionBudget configured for the Kafka brokers.

cluster-name-kafka-idx

Name given to the following Kafka resources:

- Pods created by the StrimziPodSet.
- ConfigMaps with Kafka broker configuration.

cluster-name-kafka-brokers

Service needed to have DNS resolve the Kafka broker pods IP addresses directly.

cluster-name-kafka-bootstrap

Service can be used as bootstrap servers for Kafka clients connecting from within the Kubernetes cluster.

cluster-name-kafka-external-bootstrap

Bootstrap service for clients connecting from outside the Kubernetes cluster. This resource is created only when an external listener is enabled. The old service name will be used for backwards compatibility when the listener name is `external` and port is `9094`.

cluster-name-kafka-pod-id

Service used to route traffic from outside the Kubernetes cluster to individual pods. This resource is created only when an external listener is enabled. The old service name will be used for backwards compatibility when the listener name is `external` and port is `9094`.

cluster-name-kafka-external-bootstrap

Bootstrap route for clients connecting from outside the Kubernetes cluster. This resource is created only when an external listener is enabled and set to type `route`. The old route name will be used for backwards compatibility when the listener name is `external` and port is `9094`.

cluster-name-kafka-pod-id

Route for traffic from outside the Kubernetes cluster to individual pods. This resource is created only when an external listener is enabled and set to type `route`. The old route name will be used for backwards compatibility when the listener name is `external` and port is `9094`.

cluster-name-kafka-listener-name-bootstrap

Bootstrap service for clients connecting from outside the Kubernetes cluster. This resource is

created only when an external listener is enabled. The new service name will be used for all other external listeners.

`cluster-name-kafka-listener-name-pod-id`

Service used to route traffic from outside the Kubernetes cluster to individual pods. This resource is created only when an external listener is enabled. The new service name will be used for all other external listeners.

`cluster-name-kafka-listener-name-bootstrap`

Bootstrap route for clients connecting from outside the Kubernetes cluster. This resource is created only when an external listener is enabled and set to type `route`. The new route name will be used for all other external listeners.

`cluster-name-kafka-listener-name-pod-id`

Route for traffic from outside the Kubernetes cluster to individual pods. This resource is created only when an external listener is enabled and set to type `route`. The new route name will be used for all other external listeners.

`cluster-name-kafka-config`

ConfigMap containing the Kafka ancillary configuration, which is mounted as a volume by the broker pods when the `UseStrimziPodSets` feature gate is disabled.

`cluster-name-kafka-brokers`

Secret with Kafka broker keys.

`cluster-name-network-policy-kafka`

Network policy managing access to the Kafka services.

`strimzi-namespace-name-cluster-name-kafka-init`

Cluster role binding used by the Kafka brokers.

`cluster-name-jmx`

Secret with JMX username and password used to secure the Kafka broker port. This resource is created only when JMX is enabled in Kafka.

`data-cluster-name-kafka-idx`

Persistent Volume Claim for the volume used for storing data for the Kafka broker pod `idx`. This resource is created only if persistent storage is selected for provisioning persistent volumes to store data.

`data-id-cluster-name-kafka-idx`

Persistent Volume Claim for the volume `id` used for storing data for the Kafka broker pod `idx`. This resource is created only if persistent storage is selected for JBOD volumes when provisioning persistent volumes to store data.

Entity Operator

These resources are only created if the Entity Operator is deployed using the Cluster Operator.

`cluster-name-entity-operator`

Name given to the following Entity Operator resources:

- Deployment with Topic and User Operators.
- Service account used by the Entity Operator.
- Network policy managing access to the Entity Operator metrics.

`cluster-name-entity-operator-random-string`

Pod created by the Entity Operator deployment.

`cluster-name-entity-topic-operator-config`

ConfigMap with ancillary configuration for Topic Operators.

`cluster-name-entity-user-operator-config`

ConfigMap with ancillary configuration for User Operators.

`cluster-name-entity-topic-operator-certs`

Secret with Topic Operator keys for communication with Kafka and ZooKeeper.

`cluster-name-entity-user-operator-certs`

Secret with User Operator keys for communication with Kafka and ZooKeeper.

`strimzi-cluster-name-entity-topic-operator`

Role binding used by the Entity Topic Operator.

`strimzi-cluster-name-entity-user-operator`

Role binding used by the Entity User Operator.

Kafka Exporter

These resources are only created if the Kafka Exporter is deployed using the Cluster Operator.

`cluster-name-kafka-exporter`

Name given to the following Kafka Exporter resources:

- Deployment with Kafka Exporter.
- Service used to collect consumer lag metrics.
- Service account used by the Kafka Exporter.
- Network policy managing access to the Kafka Exporter metrics.

`cluster-name-kafka-exporter-random-string`

Pod created by the Kafka Exporter deployment.

Cruise Control

These resources are only created if Cruise Control was deployed using the Cluster Operator.

`cluster-name-cruise-control`

Name given to the following Cruise Control resources:

- Deployment with Cruise Control.
- Service used to communicate with Cruise Control.
- Service account used by the Cruise Control.

`cluster-name-cruise-control-random-string`

Pod created by the Cruise Control deployment.

`cluster-name-cruise-control-config`

ConfigMap that contains the Cruise Control ancillary configuration, and is mounted as a volume by the Cruise Control pods.

`cluster-name-cruise-control-certs`

Secret with Cruise Control keys for communication with Kafka and ZooKeeper.

`cluster-name-network-policy-cruise-control`

Network policy managing access to the Cruise Control service.

2.2. Kafka Connect cluster configuration

Configure a Kafka Connect deployment using the `KafkaConnect` resource. Kafka Connect is an integration toolkit for streaming data between Kafka brokers and other systems using connector plugins. Kafka Connect provides a framework for integrating Kafka with an external data source or target, such as a database, for import or export of data using connectors. Connectors are plugins that provide the connection configuration needed.

`KafkaConnect` schema reference describes the full schema of the `KafkaConnect` resource.

For more information on deploying connector plugins, see Extending Kafka Connect with connector plugins.

2.2.1. Configuring Kafka Connect

Use Kafka Connect to set up external data connections to your Kafka cluster. Use the properties of the `KafkaConnect` resource to configure your Kafka Connect deployment.

KafkaConnector configuration

`KafkaConnector` resources allow you to create and manage connector instances for Kafka Connect in a Kubernetes-native way.

In your Kafka Connect configuration, you enable KafkaConnectors for a Kafka Connect cluster by adding the `strimzi.io/use-connector-resources` annotation. You can also add a `build` configuration so that Strimzi automatically builds a container image with the connector plugins you require for your data connections. External configuration for Kafka Connect connectors is specified through the `externalConfiguration` property.

To manage connectors, you can use use `KafkaConnector` custom resources or the Kafka Connect REST API. `KafkaConnector` resources must be deployed to the same namespace as the Kafka Connect cluster they link to. For more information on using these methods to create, reconfigure, or delete

connectors, see Adding connectors.

Connector configuration is passed to Kafka Connect as part of an HTTP request and stored within Kafka itself. ConfigMaps and Secrets are standard Kubernetes resources used for storing configurations and confidential data. You can use ConfigMaps and Secrets to configure certain elements of a connector. You can then reference the configuration values in HTTP REST commands, which keeps the configuration separate and more secure, if needed. This method applies especially to confidential data, such as usernames, passwords, or certificates.

Handling high volumes of messages

You can tune the configuration to handle high volumes of messages. For more information, see Handling high volumes of messages.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

See the *Deploying and Upgrading Strimzi* guide for instructions on running a:

- Cluster Operator
- Kafka cluster

Procedure

1. Edit the `spec` properties of the `KafkaConnect` resource.

   The properties you can configure are shown in this example configuration:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect ①
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true" ②
spec:
  replicas: 3 ③
  authentication: ④
    type: tls
    certificateAndKey:
      certificate: source.crt
      key: source.key
      secretName: my-user-source
  bootstrapServers: my-cluster-kafka-bootstrap:9092 ⑤
  tls: ⑥
    trustedCertificates:
      - secretName: my-cluster-cluster-cert
        certificate: ca.crt
      - secretName: my-cluster-cluster-cert
        certificate: ca2.crt
  config: ⑦
```

```
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
  build: ⑧
    output: ⑨
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins: ⑩
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://repo1.maven.org/maven2/io/debezium/debezium-connector-
postgres/2.1.3.Final/debezium-connector-postgres-2.1.3.Final-plugin.tar.gz
            sha512sum:
c4ddc97846de561755dc0b021a62aba656098829c70eb3ade3b817ce06d852ca12ae50c0281cc791a5a
131cb7fc21fb15f4b8ee76c6cae5dd07f9c11cb7c6e79
      - name: camel-telegram
        artifacts:
          - type: tgz
            url:
https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-
telegram-kafka-connector/0.11.5/camel-telegram-kafka-connector-0.11.5-
package.tar.gz
            sha512sum:
d6d9f45e0d1dbfcc9f6d1c7ca2046168c764389c78bc4b867dab32d24f710bb74ccf2a007d7d7a8af2d
fca09d9a52ccbc2831fc715c195a3634cca055185bd91
  externalConfiguration: ⑪
    env:
      - name: AWS_ACCESS_KEY_ID
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsAccessKey
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsSecretAccessKey
  resources: ⑫
    requests:
      cpu: "1"
      memory: 2Gi
```

```
    limits:
      cpu: "2"
      memory: 2Gi
  logging: ⑬
    type: inline
    loggers:
      log4j.rootLogger: "INFO"
  readinessProbe: ⑭
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  metricsConfig: ⑮
    type: jmxPrometheusExporter
    valueFrom:
      configMapKeyRef:
        name: my-config-map
        key: my-key
  jvmOptions: ⑯
    "-Xmx": "1g"
    "-Xms": "1g"
  image: my-org/my-image:latest ⑰
  rack:
    topologyKey: topology.kubernetes.io/zone ⑱
  template: ⑲
    pod:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: application
                    operator: In
                    values:
                      - postgresql
                      - mongodb
              topologyKey: "kubernetes.io/hostname"
    connectContainer: ⑳
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing:
    type: opentelemetry
```

① Use `KafkaConnect`.

② Enables KafkaConnectors for the Kafka Connect cluster.

③ The number of replica nodes for the workers that run tasks.

④ Authentication for the Kafka Connect cluster, specified as mTLS, token-based OAuth, SASL-based SCRAM-SHA-256/SCRAM-SHA-512, or PLAIN. By default, Kafka Connect connects to Kafka brokers using a plain text connection.

⑤ Bootstrap server for connection to the Kafka Connect cluster.

⑥ TLS encryption with key names under which TLS certificates are stored in X.509 format for the cluster. If certificates are stored in the same secret, it can be listed multiple times.

⑦ Kafka Connect configuration of workers (not connectors). Standard Apache Kafka configuration may be provided, restricted to those properties not managed directly by Strimzi.

⑧ Build configuration properties for building a container image with connector plugins automatically.

⑨ (Required) Configuration of the container registry where new images are pushed.

⑩ (Required) List of connector plugins and their artifacts to add to the new container image. Each plugin must be configured with at least one `artifact`.

⑪ External configuration for connectors using environment variables, as shown here, or volumes. You can also use configuration provider plugins to load configuration values from external sources.

⑫ Requests for reservation of supported resources, currently `cpu` and `memory`, and limits to specify the maximum resources that can be consumed.

⑬ Specified Kafka Connect loggers and log levels added directly (`inline`) or indirectly (`external`) through a ConfigMap. A custom ConfigMap must be placed under the `log4j.properties` or `log4j2.properties` key. For the Kafka Connect `log4j.rootLogger` logger, you can set the log level to INFO, ERROR, WARN, TRACE, DEBUG, FATAL or OFF.

⑭ Healthchecks to know when to restart a container (liveness) and when a container can accept traffic (readiness).

⑮ Prometheus metrics, which are enabled by referencing a ConfigMap containing configuration for the Prometheus JMX exporter in this example. You can enable metrics without further configuration using a reference to a ConfigMap containing an empty file under `metricsConfig.valueFrom.configMapKeyRef.key`.

⑯ JVM configuration options to optimize performance for the Virtual Machine (VM) running Kafka Connect.

⑰ ADVANCED OPTION: Container image configuration, which is recommended only in special situations.

⑱ SPECIALIZED OPTION: Rack awareness configuration for the deployment. This is a specialized option intended for a deployment within the same location, not across regions. Use this option if you want connectors to consume from the closest replica rather than the leader replica. In certain cases, consuming from the closest replica can improve network utilization or reduce costs . The `topologyKey` must match a node label containing the rack ID. The example used in this configuration specifies a zone using the standard `topology.kubernetes.io/zone` label. To consume from the closest replica, enable the `RackAwareReplicaSelector` in the Kafka broker configuration.

⑲ Template customization. Here a pod is scheduled with anti-affinity, so the pod is not scheduled on nodes with the same hostname.

⑳ Environment variables are set for distributed tracing.

Distributed tracing is enabled by using OpenTelemetry.

2. Create or update the resource:

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

3. If authorization is enabled for Kafka Connect, configure Kafka Connect users to enable access to the Kafka Connect consumer group and topics.

Additional resources

- Introducing distributed tracing

2.2.2. Configuring Kafka Connect for multiple instances

If you are running multiple instances of Kafka Connect, you have to change the default configuration of the following `config` properties:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: connect-cluster ①
    offset.storage.topic: connect-cluster-offsets ②
    config.storage.topic: connect-cluster-configs ③
    status.storage.topic: connect-cluster-status  ④
    # ...
# ...
```

① The Kafka Connect cluster ID within Kafka.

② Kafka topic that stores connector offsets.

③ Kafka topic that stores connector and task status configurations.

④ Kafka topic that stores connector and task status updates.

NOTE Values for the three topics must be the same for all Kafka Connect instances with the same `group.id`.

Unless you change the default settings, each Kafka Connect instance connecting to the same Kafka cluster is deployed with the same values. What happens, in effect, is all instances are coupled to run in a cluster and use the same topics.

If multiple Kafka Connect clusters try to use the same topics, Kafka Connect will not work as expected and generate errors.

If you wish to run multiple Kafka Connect instances, change the values of these properties for each instance.

2.2.3. Configuring Kafka Connect user authorization

This procedure describes how to authorize user access to Kafka Connect.

When any type of authorization is being used in Kafka, a Kafka Connect user requires read/write access rights to the consumer group and the internal topics of Kafka Connect.

The properties for the consumer group and internal topics are automatically configured by Strimzi, or they can be specified explicitly in the `spec` of the `KafkaConnect` resource.

Example configuration properties in the `KafkaConnect` resource

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster ①
    offset.storage.topic: my-connect-cluster-offsets ②
    config.storage.topic: my-connect-cluster-configs ③
    status.storage.topic: my-connect-cluster-status ④
    # ...
  # ...
```

① The Kafka Connect cluster ID within Kafka.

② Kafka topic that stores connector offsets.

③ Kafka topic that stores connector and task status configurations.

④ Kafka topic that stores connector and task status updates.

This procedure shows how access is provided when `simple` authorization is being used.

Simple authorization uses ACL rules, handled by the Kafka `AclAuthorizer` plugin, to provide the right level of access. For more information on configuring a `KafkaUser` resource to use simple authorization, see the `AclRule` schema reference.

NOTE The default values for the consumer group and topics will differ when running multiple instances.

Prerequisites

- A Kubernetes cluster

- A running Cluster Operator

Procedure

1. Edit the `authorization` property in the `KafkaUser` resource to provide access rights to the user.

   In the following example, access rights are configured for the Kafka Connect topics and consumer group using `literal` name values:

Property	Name
`offset.storage.topic`	`connect-cluster-offsets`
`status.storage.topic`	`connect-cluster-status`
`config.storage.topic`	`connect-cluster-configs`
`group`	`connect-cluster`

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  # ...
  authorization:
    type: simple
    acls:
      # access to offset.storage.topic
      - resource:
          type: topic
          name: connect-cluster-offsets
          patternType: literal
        operations:
          - Create
          - Describe
          - Read
          - Write
        host: "*"
      # access to status.storage.topic
      - resource:
          type: topic
          name: connect-cluster-status
          patternType: literal
        operations:
          - Create
          - Describe
          - Read
          - Write
        host: "*"
      # access to config.storage.topic
      - resource:
```

```
          type: topic
          name: connect-cluster-configs
          patternType: literal
        operations:
          - Create
          - Describe
          - Read
          - Write
        host: "*"
      # consumer group
      - resource:
          type: group
          name: connect-cluster
          patternType: literal
        operations:
          - Read
        host: "*"
```

2. Create or update the resource.

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

2.2.4. List of Kafka Connect cluster resources

The following resources are created by the Cluster Operator in the Kubernetes cluster:

***connect-cluster-name*-connect**

Name given to the following Kafka Connect resources:

- Deployment that creates the Kafka Connect worker node pods (when `StableConnectIdentities` feature gate is disabled).
- StrimziPodSet that creates the Kafka Connect worker node pods (when `StableConnectIdentities` feature gate is enabled).
- Headless service that provides stable DNS names to the Connect pods (when `StableConnectIdentities` feature gate is enabled).
- Pod Disruption Budget configured for the Kafka Connect worker nodes.

connect-cluster-name*-connect-*idx

Pods created by the Kafka Connect StrimziPodSet (when `StableConnectIdentities` feature gate is enabled).

***connect-cluster-name*-connect-api**

Service which exposes the REST interface for managing the Kafka Connect cluster.

***connect-cluster-name*-config**

ConfigMap which contains the Kafka Connect ancillary configuration and is mounted as a volume by the Kafka broker pods.

2.3. Kafka MirrorMaker 2 cluster configuration

Configure a Kafka MirrorMaker 2 deployment using the `KafkaMirrorMaker2` resource. MirrorMaker 2 replicates data between two or more Kafka clusters, within or across data centers.

`KafkaMirrorMaker2` schema reference describes the full schema of the `KafkaMirrorMaker2` resource.

MirrorMaker 2 resource configuration differs from the previous version of MirrorMaker. If you choose to use MirrorMaker 2, there is currently no legacy support, so any resources must be manually converted into the new format.

2.3.1. Replicating data using MirrorMaker 2

Data replication across clusters supports scenarios that require:

- Recovery of data in the event of a system failure
- Aggregation of data for analysis
- Restriction of data access to a specific cluster
- Provision of data at a specific location to improve latency

MirrorMaker 2 configuration

MirrorMaker 2 consumes messages from a source Kafka cluster and writes them to a target Kafka cluster.

MirrorMaker 2 uses:

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

MirrorMaker 2 is based on the Kafka Connect framework, *connectors* managing the transfer of data between clusters.

You configure MirrorMaker 2 to define the Kafka Connect deployment, including the connection details of the source and target clusters, and then run a set of MirrorMaker 2 connectors to make the connection.

MirrorMaker 2 consists of the following connectors:

`MirrorSourceConnector`
: The source connector replicates topics from a source cluster to a target cluster. It also replicates ACLs and is necessary for the `MirrorCheckpointConnector` to run.

`MirrorCheckpointConnector`
: The checkpoint connector periodically tracks offsets. If enabled, it also synchronizes consumer group offsets between the source and target cluster.

`MirrorHeartbeatConnector`

The heartbeat connector periodically checks connectivity between the source and target cluster.

NOTE If you are using the User Operator to manage ACLs, ACL replication through the connector is not possible.

The process of *mirroring* data from a source cluster to a target cluster is asynchronous. Each MirrorMaker 2 instance mirrors data from one source cluster to one target cluster. You can use more than one MirrorMaker 2 instance to mirror data between any number of clusters.

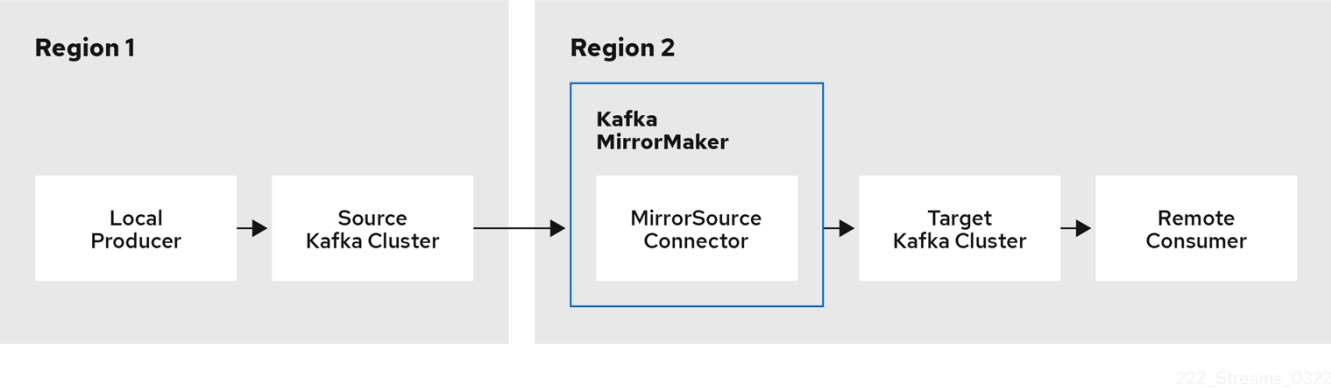


Figure 1. Replication across two clusters

By default, a check for new topics in the source cluster is made every 10 minutes. You can change the frequency by adding `refresh.topics.interval.seconds` to the source connector configuration.

Cluster configuration

You can use MirrorMaker 2 in *active/passive* or *active/active* cluster configurations.

active/active cluster configuration

An active/active configuration has two active clusters replicating data bidirectionally. Applications can use either cluster. Each cluster can provide the same data. In this way, you can make the same data available in different geographical locations. As consumer groups are active in both clusters, consumer offsets for replicated topics are not synchronized back to the source cluster.

active/passive cluster configuration

An active/passive configuration has an active cluster replicating data to a passive cluster. The passive cluster remains on standby. You might use the passive cluster for data recovery in the event of system failure.

The expectation is that producers and consumers connect to active clusters only. A MirrorMaker 2 cluster is required at each target destination.

Bidirectional replication (active/active)

The MirrorMaker 2 architecture supports bidirectional replication in an *active/active* cluster configuration.

Each cluster replicates the data of the other cluster using the concept of *source* and *remote* topics. As the same topics are stored in each cluster, remote topics are automatically renamed by MirrorMaker 2 to represent the source cluster. The name of the originating cluster is prepended to the name of the topic.

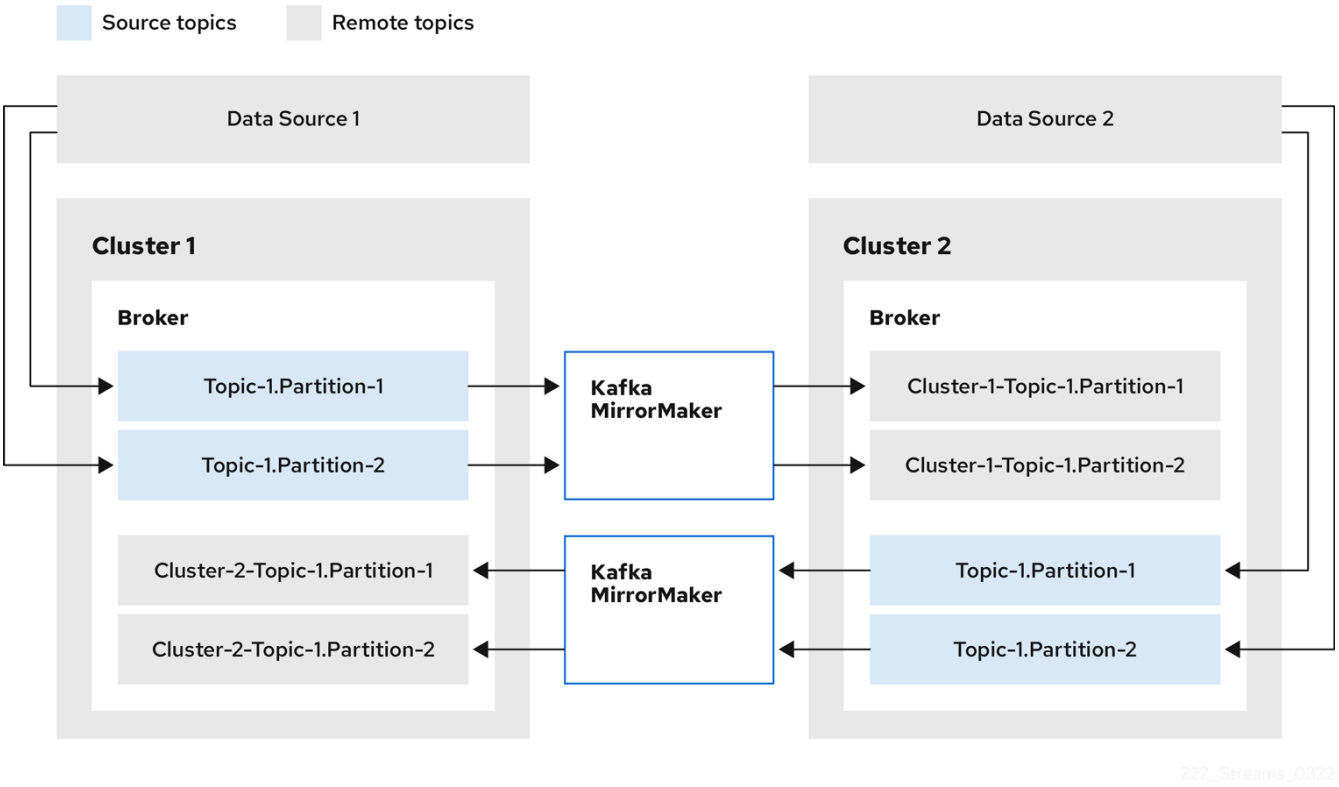


Figure 2. Topic renaming

By flagging the originating cluster, topics are not replicated back to that cluster.

The concept of replication through *remote* topics is useful when configuring an architecture that requires data aggregation. Consumers can subscribe to source and remote topics within the same cluster, without the need for a separate aggregation cluster.

Unidirectional replication (active/passive)

The MirrorMaker 2 architecture supports unidirectional replication in an *active/passive* cluster configuration.

You can use an *active/passive* cluster configuration to make backups or migrate data to another cluster. In this situation, you might not want automatic renaming of remote topics.

You can override automatic renaming by adding `IdentityReplicationPolicy` to the source connector configuration. With this configuration applied, topics retain their original names.

Topic configuration synchronization

MirrorMaker 2 supports topic configuration synchronization between source and target clusters. You specify source topics in the MirrorMaker 2 configuration. MirrorMaker 2 monitors the source topics. MirrorMaker 2 detects and propagates changes to the source topics to the remote topics.

Changes might include automatically creating missing topics and partitions.

NOTE	In most cases you write to local topics and read from remote topics. Though write operations are not prevented on remote topics, they should be avoided.

Offset tracking

MirrorMaker 2 tracks offsets for consumer groups using internal topics.

`offset-syncs` topic

The `offset-syncs` topic maps the source and target offsets for replicated topic partitions from record metadata.

`checkpoints` topic

The `checkpoints` topic maps the last committed offset in the source and target cluster for replicated topic partitions in each consumer group.

As they used internally by MirrorMaker 2, you do not interact directly with these topics.

`MirrorCheckpointConnector` emits *checkpoints* for offset tracking. Offsets for the `checkpoints` topic are tracked at predetermined intervals through configuration. Both topics enable replication to be fully restored from the correct offset position on failover.

The location of the `offset-syncs` topic is the `source` cluster by default. You can use the `offset-syncs.topic.location` connector configuration to change this to the `target` cluster. You need read/write access to the cluster that contains the topic. Using the target cluster as the location of the `offset-syncs` topic allows you to use MirrorMaker 2 even if you have only read access to the source cluster.

Synchronizing consumer group offsets

The `__consumer_offsets` topic stores information on committed offsets for each consumer group. Offset synchronization periodically transfers the consumer offsets for the consumer groups of a source cluster into the consumer offsets topic of a target cluster.

Offset synchronization is particularly useful in an *active/passive* configuration. If the active cluster goes down, consumer applications can switch to the passive (standby) cluster and pick up from the last transferred offset position.

To use topic offset synchronization, enable the synchronization by adding `sync.group.offsets.enabled` to the checkpoint connector configuration, and setting the property to `true`. Synchronization is disabled by default.

When using the `IdentityReplicationPolicy` in the source connector, it also has to be configured in the checkpoint connector configuration. This ensures that the mirrored consumer offsets will be applied for the correct topics.

Consumer offsets are only synchronized for consumer groups that are not active in the target cluster. If the consumer groups are in the target cluster, the synchronization cannot be performed and an `UNKNOWN_MEMBER_ID` error is returned.

If enabled, the synchronization of offsets from the source cluster is made periodically. You can change the frequency by adding `sync.group.offsets.interval.seconds` and `emit.checkpoints.interval.seconds` to the checkpoint connector configuration. The properties specify the frequency in seconds that the consumer group offsets are synchronized, and the frequency of checkpoints emitted for offset tracking. The default for both properties is 60 seconds. You can also change the frequency of checks for new consumer groups using the `refresh.groups.interval.seconds` property, which is performed every 10 minutes by default.

Because the synchronization is time-based, any switchover by consumers to a passive cluster will likely result in some duplication of messages.

NOTE	If you have an application written in Java, you can use the `RemoteClusterUtils.java` utility to synchronize offsets through the application. The utility fetches remote offsets for a consumer group from the `checkpoints` topic.

Deciding when to use the heartbeat connector

The heartbeat connector emits heartbeats to check connectivity between source and target Kafka clusters. An internal `heartbeat` topic is replicated from the source cluster, which means that the heartbeat connector must be connected to the source cluster. The `heartbeat` topic is located on the target cluster, which allows it to do the following:

- Identify all source clusters it is mirroring data from
- Verify the liveness and latency of the mirroring process

This helps to make sure that the process is not stuck or has stopped for any reason. While the heartbeat connector can be a valuable tool for monitoring the mirroring processes between Kafka clusters, it's not always necessary to use it. For example, if your deployment has low network latency or a small number of topics, you might prefer to monitor the mirroring process using log messages or other monitoring tools. If you decide not to use the heartbeat connector, simply omit it from your MirrorMaker 2 configuration.

2.3.2. Connector configuration

Use Mirrormaker 2 connector configuration for the internal connectors that orchestrate the synchronization of data between Kafka clusters.

The following table describes connector properties and the connectors you configure to use them.

Table 2. MirrorMaker 2 connector configuration properties

Property	sourceConnector	checkpointConnector	heartbeatConnector
admin.timeout.ms Timeout for admin tasks, such as detecting new topics. Default is `60000` (1 minute).	□	□	□

Property	sourceConnector	checkpointConnector	heartbeatConnector
replication.policy.class Policy to define the remote topic naming convention. Default is `org.apache.kafka.connect.mirror.DefaultReplicationPolicy`.	✓	✓	✓
replication.policy.separator The separator used for topic naming in the target cluster. Default is `.` (dot).	✓	✓	✓
consumer.poll.timeout.ms Timeout when polling the source cluster. Default is `1000` (1 second).	✓	✓	
offset-syncs.topic.location The location of the `offset-syncs` topic, which can be the `source` (default) or `target` cluster.	✓	✓	
topic.filter.class Topic filter to select the topics to replicate. Default is `org.apache.kafka.connect.mirror.DefaultTopicFilter`.	✓	✓	
config.property.filter.class Topic filter to select the topic configuration properties to replicate. Default is `org.apache.kafka.connect.mirror.DefaultConfigPropertyFilter`.	✓		
config.properties.exclude Topic configuration properties that should not be replicated. Supports comma-separated property names and regular expressions.	✓		
offset.lag.max Maximum allowable (out-of-sync) offset lag before a remote partition is synchronized. Default is `100`.	✓		
offset-syncs.topic.replication.factor Replication factor for the internal `offset-syncs` topic. Default is `3`.	✓		

Property	sourceConnector	checkpointConnector	heartbeatConnector
refresh.topics.enabled Enables check for new topics and partitions. Default is `true`.	✓		
refresh.topics.interval.seconds Frequency of topic refresh. Default is `600` (10 minutes).	✓		
replication.factor The replication factor for new topics. Default is `2`.	✓		
sync.topic.acls.enabled Enables synchronization of ACLs from the source cluster. Default is `true`. Not compatible with the User Operator.	✓		
sync.topic.acls.interval.seconds Frequency of ACL synchronization. Default is `600` (10 minutes).	✓		
sync.topic.configs.enabled Enables synchronization of topic configuration from the source cluster. Default is `true`.	✓		
sync.topic.configs.interval.seconds Frequency of topic configuration synchronization. Default `600` (10 minutes).	✓		
checkpoints.topic.replication.factor Replication factor for the internal `checkpoints` topic. Default is `3`.		✓	
emit.checkpoints.enabled Enables synchronization of consumer offsets to the target cluster. Default is `true`.		✓	
emit.checkpoints.interval.seconds Frequency of consumer offset synchronization. Default is `60` (1 minute).		✓	

Property	sourceConnector	checkpointConnector	heartbeatConnector
group.filter.class Group filter to select the consumer groups to replicate. Default is `org.apache.kafka.connect.mirror.DefaultGroupFilter`.		□	
refresh.groups.enabled Enables check for new consumer groups. Default is `true`.		□	
refresh.groups.interval.seconds Frequency of consumer group refresh. Default is `600` (10 minutes).		□	
sync.group.offsets.enabled Enables synchronization of consumer group offsets to the target cluster `__consumer_offsets` topic. Default is `false`.		□	
sync.group.offsets.interval.seconds Frequency of consumer group offset synchronization. Default is `60` (1 minute).		□	
emit.heartbeats.enabled Enables connectivity checks on the target cluster. Default is `true`.			□
emit.heartbeats.interval.seconds Frequency of connectivity checks. Default is `1` (1 second).			□
heartbeats.topic.replication.factor Replication factor for the internal `heartbeats` topic. Default is `3`.			□

2.3.3. Connector producer and consumer configuration

MirrorMaker 2 connectors use internal producers and consumers. If needed, you can configure these producers and consumers to override the default settings.

For example, you can increase the `batch.size` for the source producer that sends topics to the target Kafka cluster to better accommodate large volumes of messages.

IMPORTANT	Producer and consumer configuration options depend on the MirrorMaker 2 implementation, and may be subject to change.

The following tables describe the producers and consumers for each of the connectors and where you can add configuration.

Table 3. Source connector producers and consumers

Type	Description	Configuration
Producer	Sends topic messages to the target Kafka cluster. Consider tuning the configuration of this producer when it is handling large volumes of data.	`mirrors.sourceConnector.config: producer.override.*`
Producer	Writes to the `offset-syncs` topic, which maps the source and target offsets for replicated topic partitions.	`mirrors.sourceConnector.config: producer.*`
Consumer	Retrieves topic messages from the source Kafka cluster.	`mirrors.sourceConnector.config: consumer.*`

Table 4. Checkpoint connector producers and consumers

Type	Description	Configuration
Producer	Emits consumer offset checkpoints.	`mirrors.checkpointConnector.config: producer.override.*`
Consumer	Loads the `offset-syncs` topic.	`mirrors.checkpointConnector.config: consumer.*`

NOTE You can set `offset-syncs.topic.location` to `target` to use the target Kafka cluster as the location of the `offset-syncs` topic.

Table 5. Heartbeat connector producer

Type	Description	Configuration
Producer	Emits heartbeats.	`mirrors.heartbeatConnector.config: producer.override.*`

The following example shows how you configure the producers and consumers.

Example configuration for connector producers and consumers

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 3.4.0
  # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      tasksMax: 5
      config:
        producer.override.batch.size: 327680
        producer.override.linger.ms: 100
        producer.request.timeout.ms: 30000
        consumer.fetch.max.bytes: 52428800
        # ...
    checkpointConnector:
      config:
        producer.override.request.timeout.ms: 30000
        consumer.max.poll.interval.ms: 300000
        # ...
    heartbeatConnector:
      config:
        producer.override.request.timeout.ms: 30000
        # ...
```

Additional resources

- `KafkaMirrorMaker2ConnectorSpec` schema reference
- `KafkaMirrorMaker2MirrorSpec` schema reference

2.3.4. Specifying a maximum number of tasks

Connectors create the tasks that are responsible for moving data in and out of Kafka. Each connector comprises one or more tasks that are distributed across a group of worker pods that run the tasks. Increasing the number of tasks can help with performance issues when replicating a large number of partitions or synchronizing the offsets of a large number of consumer groups.

Tasks run in parallel. Workers are assigned one or more tasks. A single task is handled by one worker pod, so you don't need more worker pods than tasks. If there are more tasks than workers, workers handle multiple tasks.

You can specify the maximum number of connector tasks in your MirrorMaker configuration using the `tasksMax` property. Without specifying a maximum number of tasks, the default setting is a single task.

The heartbeat connector always uses a single task.

The number of tasks that are started for the source and checkpoint connectors is the lower value between the maximum number of possible tasks and the value for `tasksMax`. For the source connector, the maximum number of tasks possible is one for each partition being replicated from the source cluster. For the checkpoint connector, the maximum number of tasks possible is one for each consumer group being replicated from the source cluster. When setting a maximum number of tasks, consider the number of partitions and the hardware resources that support the process.

If the infrastructure supports the processing overhead, increasing the number of tasks can improve throughput and latency. For example, adding more tasks reduces the time taken to poll the source cluster when there is a high number of partitions or consumer groups.

Increasing the number of tasks for the source connector is useful when you have a large number of partitions.

Increasing the number of tasks for the source connector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector:
      tasksMax: 10
  # ...
```

Increasing the number of tasks for the checkpoint connector is useful when you have a large number of consumer groups.

Increasing the number of tasks for the checkpoint connector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  # ...
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    checkpointConnector:
      tasksMax: 10
  # ...
```

By default, MirrorMaker 2 checks for new consumer groups every 10 minutes. You can adjust the

`refresh.groups.interval.seconds` configuration to change the frequency. Take care when adjusting lower. More frequent checks can have a negative impact on performance.

Checking connector task operations

If you are using Prometheus and Grafana to monitor your deployment, you can check MirrorMaker 2 performance. The example MirrorMaker 2 Grafana dashboard provided with Strimzi shows the following metrics related to tasks and latency.

- The number of tasks
- Replication latency
- Offset synchronization latency

Additional resources

- Grafana dashboards

2.3.5. ACL rules synchronization

ACL access to remote topics is possible if you are **not** using the User Operator.

If `AclAuthorizer` is being used, without the User Operator, ACL rules that manage access to brokers also apply to remote topics. Users that can read a source topic can read its remote equivalent.

NOTE	OAuth 2.0 authorization does not support access to remote topics in this way.

2.3.6. Configuring Kafka MirrorMaker 2

Use the properties of the `KafkaMirrorMaker2` resource to configure your Kafka MirrorMaker 2 deployment. Use MirrorMaker 2 to synchronize data between Kafka clusters.

The configuration must specify:

- Each Kafka cluster
- Connection information for each cluster, including authentication
- The replication flow and direction
  - Cluster to cluster
  - Topic to topic

NOTE	The previous version of MirrorMaker continues to be supported. If you wish to use the resources configured for the previous version, they must be updated to the format supported by MirrorMaker 2.

MirrorMaker 2 provides default configuration values for properties such as replication factors. A minimal configuration, with defaults left unchanged, would be something like this example:

Minimal configuration for MirrorMaker 2

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
spec:
  version: 3.4.0
  connectCluster: "my-cluster-target"
  clusters:
  - alias: "my-cluster-source"
    bootstrapServers: my-cluster-source-kafka-bootstrap:9092
  - alias: "my-cluster-target"
    bootstrapServers: my-cluster-target-kafka-bootstrap:9092
  mirrors:
  - sourceCluster: "my-cluster-source"
    targetCluster: "my-cluster-target"
    sourceConnector: {}
```

You can configure access control for source and target clusters using mTLS or SASL authentication. This procedure shows a configuration that uses TLS encryption and mTLS authentication for the source and target cluster.

You can specify the topics and consumer groups you wish to replicate from a source cluster in the `KafkaMirrorMaker2` resource. You use the `topicsPattern` and `groupsPattern` properties to do this. You can provide a list of names or use a regular expression. By default, all topics and consumer groups are replicated if you do not set the `topicsPattern` and `groupsPattern` properties. You can also replicate all topics and consumer groups by using `".*"` as a regular expression. However, try to specify only the topics and consumer groups you need to avoid causing any unnecessary extra load on the cluster.

Handling high volumes of messages

You can tune the configuration to handle high volumes of messages. For more information, see Handling high volumes of messages.

Prerequisites

- Strimzi is running
- Source and target Kafka clusters are available

Procedure

1. Edit the `spec` properties for the `KafkaMirrorMaker2` resource.

   The properties you can configure are shown in this example configuration:

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: KafkaMirrorMaker2
   metadata:
     name: my-mirror-maker2
   ```

```
spec:
  version: 3.4.0 ①
  replicas: 3 ②
  connectCluster: "my-cluster-target" ③
  clusters: ④
  - alias: "my-cluster-source" ⑤
    authentication: ⑥
      certificateAndKey:
        certificate: source.crt
        key: source.key
        secretName: my-user-source
      type: tls
    bootstrapServers: my-cluster-source-kafka-bootstrap:9092 ⑦
    tls: ⑧
      trustedCertificates:
      - certificate: ca.crt
        secretName: my-cluster-source-cluster-ca-cert
  - alias: "my-cluster-target" ⑨
    authentication: ⑩
      certificateAndKey:
        certificate: target.crt
        key: target.key
        secretName: my-user-target
      type: tls
    bootstrapServers: my-cluster-target-kafka-bootstrap:9092 ⑪
    config: ⑫
      config.storage.replication.factor: 1
      offset.storage.replication.factor: 1
      status.storage.replication.factor: 1
    tls: ⑬
      trustedCertificates:
      - certificate: ca.crt
        secretName: my-cluster-target-cluster-ca-cert
  mirrors: ⑭
  - sourceCluster: "my-cluster-source" ⑮
    targetCluster: "my-cluster-target" ⑯
    sourceConnector: ⑰
      tasksMax: 10 ⑱
      autoRestart: ⑲
        enabled: true
      config:
        replication.factor: 1 ⑳
        offset-syncs.topic.replication.factor: 1
        sync.topic.acls.enabled: "false"
        refresh.topics.interval.seconds: 60
        replication.policy.separator: "."
        replication.policy.class:
"org.apache.kafka.connect.mirror.IdentityReplicationPolicy"
    heartbeatConnector:
      autoRestart:
        enabled: true
```

```
      config:
        heartbeats.topic.replication.factor: 1
    checkpointConnector:
      autoRestart:
        enabled: true
      config:
        checkpoints.topic.replication.factor: 1
        refresh.groups.interval.seconds: 600
        sync.group.offsets.enabled: true
        sync.group.offsets.interval.seconds: 60
        emit.checkpoints.interval.seconds: 60
        replication.policy.class:
"org.apache.kafka.connect.mirror.IdentityReplicationPolicy"
    topicsPattern: "topic1|topic2|topic3"
    groupsPattern: "group1|group2|group3"
  resources:
    requests:
      cpu: "1"
      memory: 2Gi
    limits:
      cpu: "2"
      memory: 2Gi
  logging:
    type: inline
    loggers:
      connect.root.logger.level: "INFO"
  readinessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  jvmOptions:
    "-Xmx": "1g"
    "-Xms": "1g"
  image: my-org/my-image:latest
  rack:
    topologyKey: topology.kubernetes.io/zone
  template:
    pod:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: application
                    operator: In
                    values:
                      - postgresql
                      - mongodb
              topologyKey: "kubernetes.io/hostname"
```

```
    connectContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing:
    type: opentelemetry
  externalConfiguration:
    env:
      - name: AWS_ACCESS_KEY_ID
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsAccessKey
      - name: AWS_SECRET_ACCESS_KEY
        valueFrom:
          secretKeyRef:
            name: aws-creds
            key: awsSecretAccessKey
```

① The Kafka Connect and Mirror Maker 2.0 version, which will always be the same.

② The number of replica nodes for the workers that run tasks.

③ Kafka cluster alias for Kafka Connect, which must specify the **target** Kafka cluster. The Kafka cluster is used by Kafka Connect for its internal topics.

④ Specification for the Kafka clusters being synchronized.

⑤ Cluster alias for the source Kafka cluster.

⑥ Authentication for the source cluster, specified as mTLS, token-based OAuth, SASL-based SCRAM-SHA-256/SCRAM-SHA-512, or PLAIN.

⑦ Bootstrap server for connection to the source Kafka cluster.

⑧ TLS encryption with key names under which TLS certificates are stored in X.509 format for the source Kafka cluster. If certificates are stored in the same secret, it can be listed multiple times.

⑨ Cluster alias for the target Kafka cluster.

⑩ Authentication for the target Kafka cluster is configured in the same way as for the source Kafka cluster.

⑪ Bootstrap server for connection to the target Kafka cluster.

⑫ Kafka Connect configuration. Standard Apache Kafka configuration may be provided, restricted to those properties not managed directly by Strimzi.

⑬ TLS encryption for the target Kafka cluster is configured in the same way as for the source Kafka cluster.

⑭ MirrorMaker 2 connectors.

⑮ Cluster alias for the source cluster used by the MirrorMaker 2 connectors.

⑯ Cluster alias for the target cluster used by the MirrorMaker 2 connectors.

⑰ Configuration for the `MirrorSourceConnector` that creates remote topics. The `config` overrides the default configuration options.

⑱ The maximum number of tasks that the connector may create. Tasks handle the data replication and run in parallel. If the infrastructure supports the processing overhead, increasing this value can improve throughput. Kafka Connect distributes the tasks between members of the cluster. If there are more tasks than workers, workers are assigned multiple tasks. For sink connectors, aim to have one task for each topic partition consumed. For source connectors, the number of tasks that can run in parallel may also depend on the external system. The connector creates fewer than the maximum number of tasks if it cannot achieve the parallelism.

⑲ Enables automatic restarts of failed connectors and tasks. Up to seven restart attempts are made, after which restarts must be made manually.

⑳ Replication factor for mirrored topics created at the target cluster.

Replication factor for the `MirrorSourceConnector offset-syncs` internal topic that maps the offsets of the source and target clusters.

When ACL rules synchronization is enabled, ACLs are applied to synchronized topics. The default is `true`. This feature is not compatible with the User Operator. If you are using the User Operator, set this property to `false`.

Optional setting to change the frequency of checks for new topics. The default is for a check every 10 minutes.

Defines the separator used for the renaming of remote topics.

Adds a policy that overrides the automatic renaming of remote topics. Instead of prepending the name with the name of the source cluster, the topic retains its original name. This optional setting is useful for active/passive backups and data migration. To configure topic offset synchronization, this property must also be set for the `checkpointConnector.config`.

Configuration for the `MirrorHeartbeatConnector` that performs connectivity checks. The `config` overrides the default configuration options.

Replication factor for the heartbeat topic created at the target cluster.

Configuration for the `MirrorCheckpointConnector` that tracks offsets. The `config` overrides the default configuration options.

Replication factor for the checkpoints topic created at the target cluster.

Optional setting to change the frequency of checks for new consumer groups. The default is for a check every 10 minutes.

Optional setting to synchronize consumer group offsets, which is useful for recovery in an active/passive configuration. Synchronization is not enabled by default.

If the synchronization of consumer group offsets is enabled, you can adjust the frequency of the synchronization.

Adjusts the frequency of checks for offset tracking. If you change the frequency of offset synchronization, you might also need to adjust the frequency of these checks.

Topic replication from the source cluster defined as a comma-separated list or regular expression pattern. The source connector replicates the specified topics. The checkpoint connector tracks offsets for the specified topics. Here we request three topics by name.

Consumer group replication from the source cluster defined as a comma-separated list or regular expression pattern. The checkpoint connector replicates the specified consumer groups. Here we request three consumer groups by name.

Requests for reservation of supported resources, currently `cpu` and `memory`, and limits to specify the maximum resources that can be consumed.

Specified Kafka Connect loggers and log levels added directly (`inline`) or indirectly (`external`) through a ConfigMap. A custom ConfigMap must be placed under the `log4j.properties` or `log4j2.properties` key. For the Kafka Connect `log4j.rootLogger` logger, you can set the log level to INFO, ERROR, WARN, TRACE, DEBUG, FATAL or OFF.

Healthchecks to know when to restart a container (liveness) and when a container can accept traffic (readiness).

JVM configuration options to optimize performance for the Virtual Machine (VM) running Kafka MirrorMaker.

ADVANCED OPTION: Container image configuration, which is recommended only in special situations.

SPECIALIZED OPTION: Rack awareness configuration for the deployment. This is a specialized option intended for a deployment within the same location, not across regions. Use this option if you want connectors to consume from the closest replica rather than the leader replica. In certain cases, consuming from the closest replica can improve network utilization or reduce costs . The `topologyKey` must match a node label containing the rack ID. The example used in this configuration specifies a zone using the standard `topology.kubernetes.io/zone` label. To consume from the closest replica, enable the `RackAwareReplicaSelector` in the Kafka broker configuration.

Template customization. Here a pod is scheduled with anti-affinity, so the pod is not scheduled on nodes with the same hostname.

Environment variables are set for distributed tracing.

Distributed tracing is enabled by using OpenTelemetry.

External configuration for a Kubernetes Secret mounted to Kafka MirrorMaker as an environment variable. You can also use configuration provider plugins to load configuration values from external sources.

2. Create or update the resource:

```
kubectl apply -f MIRRORMAKER-CONFIGURATION-FILE
```

Additional resources

- Introducing distributed tracing

2.3.7. Securing a Kafka MirrorMaker 2 deployment

This procedure describes in outline the configuration required to secure a MirrorMaker 2 deployment.

You need separate configuration for the source Kafka cluster and the target Kafka cluster. You also need separate user configuration to provide the credentials required for MirrorMaker to connect to the source and target Kafka clusters.

For the Kafka clusters, you specify internal listeners for secure connections within a Kubernetes cluster and external listeners for connections outside the Kubernetes cluster.

You can configure authentication and authorization mechanisms. The security options implemented for the source and target Kafka clusters must be compatible with the security options implemented for MirrorMaker 2.

After you have created the cluster and user authentication credentials, you specify them in your MirrorMaker configuration for secure connections.

NOTE	In this procedure, the certificates generated by the Cluster Operator are used, but you can replace them by installing your own certificates. You can also configure your listener to use a Kafka listener certificate managed by an external CA (certificate authority).

Before you start

Before starting this procedure, take a look at the example configuration files provided by Strimzi. They include examples for securing a deployment of MirrorMaker 2 using mTLS or SCRAM-SHA-512 authentication. The examples specify internal listeners for connecting within a Kubernetes cluster.

The examples provide the configuration for full authorization, including all the ACLs needed by MirrorMaker 2 to allow operations on the source and target Kafka clusters.

Prerequisites

- Strimzi is running
- Separate namespaces for source and target clusters

The procedure assumes that the source and target Kafka clusters are installed to separate namespaces If you want to use the Topic Operator, you'll need to do this. The Topic Operator only watches a single cluster in a specified namespace.

By separating the clusters into namespaces, you will need to copy the cluster secrets so they can be accessed outside the namespace. You need to reference the secrets in the MirrorMaker configuration.

Procedure

1. Configure two `Kafka` resources, one to secure the source Kafka cluster and one to secure the target Kafka cluster.

   You can add listener configuration for authentication and enable authorization.

In this example, an internal listener is configured for a Kafka cluster with TLS encryption and mTLS authentication. Kafka `simple` authorization is enabled.

Example source Kafka cluster configuration with TLS encryption and mTLS authentication

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-source-cluster
spec:
  kafka:
    version: 3.4.0
    replicas: 1
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    authorization:
      type: simple
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      default.replication.factor: 1
      min.insync.replicas: 1
      inter.broker.protocol.version: "3.4"
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
  zookeeper:
    replicas: 1
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

Example target Kafka cluster configuration with TLS encryption and mTLS authentication

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

```
metadata:
  name: my-target-cluster
spec:
  kafka:
    version: 3.4.0
    replicas: 1
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    authorization:
      type: simple
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      default.replication.factor: 1
      min.insync.replicas: 1
      inter.broker.protocol.version: "3.4"
    storage:
      type: jbod
      volumes:
        - id: 0
          type: persistent-claim
          size: 100Gi
          deleteClaim: false
  zookeeper:
    replicas: 1
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

2. Create or update the `Kafka` resources in separate namespaces.

```
kubectl apply -f <kafka_configuration_file> -n <namespace>
```

The Cluster Operator creates the listeners and sets up the cluster and client certificate authority (CA) certificates to enable authentication within the Kafka cluster.

The certificates are created in the secret `<cluster_name>-cluster-ca-cert`.

3. Configure two `KafkaUser` resources, one for a user of the source Kafka cluster and one for a user

of the target Kafka cluster.

a. Configure the same authentication and authorization types as the corresponding source and target Kafka cluster. For example, if you used `tls` authentication and the `simple` authorization type in the `Kafka` configuration for the source Kafka cluster, use the same in the `KafkaUser` configuration.

b. Configure the ACLs needed by MirrorMaker 2 to allow operations on the source and target Kafka clusters.

   The ACLs are used by the internal MirrorMaker connectors, and by the underlying Kafka Connect framework.

Example source user configuration for mTLS authentication

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-source-user
  labels:
    strimzi.io/cluster: my-source-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # MirrorSourceConnector
      - resource: # Not needed if offset-syncs.topic.location=target
          type: topic
          name: mm2-offset-syncs.my-target-cluster.internal
        operations:
          - Create
          - DescribeConfigs
          - Read
          - Write
      - resource: # Needed for every topic which is mirrored
          type: topic
          name: "*"
        operations:
          - DescribeConfigs
          - Read
      # MirrorCheckpointConnector
      - resource:
          type: cluster
        operations:
          - Describe
      - resource: # Needed for every group for which offsets are synced
          type: group
          name: "*"
        operations:
```

```
          - Describe
      - resource: # Not needed if offset-syncs.topic.location=target
          type: topic
          name: mm2-offset-syncs.my-target-cluster.internal
        operations:
          - Read
```

Example target user configuration for mTLS authentication

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-target-user
  labels:
    strimzi.io/cluster: my-target-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # Underlying Kafka Connect internal topics to store configuration, offsets,
or status
      - resource:
          type: group
          name: mirrormaker2-cluster
        operations:
          - Read
      - resource:
          type: topic
          name: mirrormaker2-cluster-configs
        operations:
          - Create
          - Describe
          - DescribeConfigs
          - Read
          - Write
      - resource:
          type: topic
          name: mirrormaker2-cluster-status
        operations:
          - Create
          - Describe
          - DescribeConfigs
          - Read
          - Write
      - resource:
          type: topic
          name: mirrormaker2-cluster-offsets
        operations:
```

```
      - Create
      - Describe
      - DescribeConfigs
      - Read
      - Write
  # MirrorSourceConnector
  - resource: # Needed for every topic which is mirrored
      type: topic
      name: "*"
    operations:
      - Create
      - Alter
      - AlterConfigs
      - Write
  # MirrorCheckpointConnector
  - resource:
      type: cluster
    operations:
      - Describe
  - resource:
      type: topic
      name: my-source-cluster.checkpoints.internal
    operations:
      - Create
      - Describe
      - Read
      - Write
  - resource: # Needed for every group for which the offset is synced
      type: group
      name: "*"
    operations:
      - Read
      - Describe
  # MirrorHeartbeatConnector
  - resource:
      type: topic
      name: heartbeats
    operations:
      - Create
      - Describe
      - Write
```

NOTE You can use a certificate issued outside the User Operator by setting `type` to `tls-external`. For more information, see `KafkaUserSpec` schema reference.

4. Create or update a `KafkaUser` resource in each of the namespaces you created for the source and target Kafka clusters.

```
kubectl apply -f <kafka_user_configuration_file> -n <namespace>
```

The User Operator creates the users representing the client (MirrorMaker), and the security credentials used for client authentication, based on the chosen authentication type.

The User Operator creates a new secret with the same name as the `KafkaUser` resource. The secret contains a private and public key for mTLS authentication. The public key is contained in a user certificate, which is signed by the clients CA.

5. Configure a `KafkaMirrorMaker2` resource with the authentication details to connect to the source and target Kafka clusters.

   Example MirrorMaker 2 configuration with TLS encryption and mTLS authentication

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker-2
spec:
  version: 3.4.0
  replicas: 1
  connectCluster: "my-target-cluster"
  clusters:
    - alias: "my-source-cluster"
      bootstrapServers: my-source-cluster-kafka-bootstrap:9093
      tls: ①
        trustedCertificates:
          - secretName: my-source-cluster-cluster-ca-cert
            certificate: ca.crt
      authentication: ②
        type: tls
        certificateAndKey:
          secretName: my-source-user
          certificate: user.crt
          key: user.key
    - alias: "my-target-cluster"
      bootstrapServers: my-target-cluster-kafka-bootstrap:9093
      tls: ③
        trustedCertificates:
          - secretName: my-target-cluster-cluster-ca-cert
            certificate: ca.crt
      authentication: ④
        type: tls
        certificateAndKey:
          secretName: my-target-user
          certificate: user.crt
          key: user.key
      config:
        # -1 means it will use the default replication factor configured in the
broker
```

```
      config.storage.replication.factor: -1
      offset.storage.replication.factor: -1
      status.storage.replication.factor: -1
  mirrors:
    - sourceCluster: "my-source-cluster"
      targetCluster: "my-target-cluster"
      sourceConnector:
        config:
          replication.factor: 1
          offset-syncs.topic.replication.factor: 1
          sync.topic.acls.enabled: "false"
      heartbeatConnector:
        config:
          heartbeats.topic.replication.factor: 1
      checkpointConnector:
        config:
          checkpoints.topic.replication.factor: 1
          sync.group.offsets.enabled: "true"
      topicsPattern: "topic1|topic2|topic3"
      groupsPattern: "group1|group2|group3"
```

① The TLS certificates for the source Kafka cluster. If they are in a separate namespace, copy the cluster secrets from the namespace of the Kafka cluster.

② The user authentication for accessing the source Kafka cluster using the TLS mechanism.

③ The TLS certificates for the target Kafka cluster.

④ The user authentication for accessing the target Kafka cluster.

6. Create or update the `KafkaMirrorMaker2` resource in the same namespace as the target Kafka cluster.

```
kubectl apply -f <mirrormaker2_configuration_file> -n <namespace_of_target_cluster>
```

Additional resources

- type-KafkaMirrorMaker2ClusterSpec-reference[]

2.3.8. Performing a restart of a Kafka MirrorMaker 2 connector

This procedure describes how to manually trigger a restart of a Kafka MirrorMaker 2 connector by using a Kubernetes annotation.

Prerequisites

- The Cluster Operator is running.

Procedure

1. Find the name of the `KafkaMirrorMaker2` custom resource that controls the Kafka MirrorMaker 2 connector you want to restart:

```
kubectl get KafkaMirrorMaker2
```

2. Find the name of the Kafka MirrorMaker 2 connector to be restarted from the `KafkaMirrorMaker2` custom resource.

```
kubectl describe KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME
```

3. To restart the connector, annotate the `KafkaMirrorMaker2` resource in Kubernetes. In this example, `kubectl annotate` restarts a connector named `my-source->my-target.MirrorSourceConnector`:

```
kubectl annotate KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME "strimzi.io/restart-connector=my-source->my-target.MirrorSourceConnector"
```

4. Wait for the next reconciliation to occur (every two minutes by default).

   The Kafka MirrorMaker 2 connector is restarted, as long as the annotation was detected by the reconciliation process. When the restart request is accepted, the annotation is removed from the `KafkaMirrorMaker2` custom resource.

Additional resources

- Kafka MirrorMaker 2 cluster configuration.

2.3.9. Performing a restart of a Kafka MirrorMaker 2 connector task

This procedure describes how to manually trigger a restart of a Kafka MirrorMaker 2 connector task by using a Kubernetes annotation.

Prerequisites

- The Cluster Operator is running.

Procedure

1. Find the name of the `KafkaMirrorMaker2` custom resource that controls the Kafka MirrorMaker 2 connector you want to restart:

```
kubectl get KafkaMirrorMaker2
```

2. Find the name of the Kafka MirrorMaker 2 connector and the ID of the task to be restarted from the `KafkaMirrorMaker2` custom resource. Task IDs are non-negative integers, starting from 0.

```
kubectl describe KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME
```

3. To restart the connector task, annotate the `KafkaMirrorMaker2` resource in Kubernetes. In this example, `kubectl annotate` restarts task *0* of a connector named `my-source->my-`

`target.MirrorSourceConnector`:

```
kubectl annotate KafkaMirrorMaker2 KAFKAMIRRORMAKER-2-NAME "strimzi.io/restart-connector-task=my-source->my-target.MirrorSourceConnector:0"
```

4. Wait for the next reconciliation to occur (every two minutes by default).

   The Kafka MirrorMaker 2 connector task is restarted, as long as the annotation was detected by the reconciliation process. When the restart task request is accepted, the annotation is removed from the `KafkaMirrorMaker2` custom resource.

Additional resources

- Kafka MirrorMaker 2 cluster configuration.

2.4. Kafka MirrorMaker cluster configuration

Configure a Kafka MirrorMaker deployment using the `KafkaMirrorMaker` resource. KafkaMirrorMaker replicates data between Kafka clusters.

`KafkaMirrorMaker` schema reference describes the full schema of the `KafkaMirrorMaker` resource.

You can use Strimzi with MirrorMaker or MirrorMaker 2. MirrorMaker 2 is the latest version, and offers a more efficient way to mirror data between Kafka clusters.

IMPORTANT	Kafka MirrorMaker 1 (referred to as just *MirrorMaker* in the documentation) has been deprecated in Apache Kafka 3.0.0 and will be removed in Apache Kafka 4.0.0. As a result, the `KafkaMirrorMaker` custom resource which is used to deploy Kafka MirrorMaker 1 has been deprecated in Strimzi as well. The `KafkaMirrorMaker` resource will be removed from Strimzi when we adopt Apache Kafka 4.0.0. As a replacement, use the `KafkaMirrorMaker2` custom resource with the `IdentityReplicationPolicy`.

2.4.1. Configuring Kafka MirrorMaker

Use the properties of the `KafkaMirrorMaker` resource to configure your Kafka MirrorMaker deployment.

You can configure access control for producers and consumers using TLS or SASL authentication. This procedure shows a configuration that uses TLS encryption and mTLS authentication on the consumer and producer side.

Prerequisites

- See the *Deploying and Upgrading Strimzi* guide for instructions on running a:
  - Cluster Operator
  - Kafka cluster
- Source and target Kafka clusters must be available

Procedure

1. Edit the `spec` properties for the `KafkaMirrorMaker` resource.

   The properties you can configure are shown in this example configuration:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  replicas: 3 ①
  consumer:
    bootstrapServers: my-source-cluster-kafka-bootstrap:9092 ②
    groupId: "my-group" ③
    numStreams: 2 ④
    offsetCommitInterval: 120000 ⑤
    tls: ⑥
      trustedCertificates:
      - secretName: my-source-cluster-ca-cert
        certificate: ca.crt
    authentication: ⑦
      type: tls
      certificateAndKey:
        secretName: my-source-secret
        certificate: public.crt
        key: private.key
    config: ⑧
      max.poll.records: 100
      receive.buffer.bytes: 32768
  producer:
    bootstrapServers: my-target-cluster-kafka-bootstrap:9092
    abortOnSendFailure: false ⑨
    tls:
      trustedCertificates:
      - secretName: my-target-cluster-ca-cert
        certificate: ca.crt
    authentication:
      type: tls
      certificateAndKey:
        secretName: my-target-secret
        certificate: public.crt
        key: private.key
    config:
      compression.type: gzip
      batch.size: 8192
  include: "my-topic|other-topic" ⑩
  resources: ⑪
    requests:
      cpu: "1"
      memory: 2Gi
```

```
    limits:
      cpu: "2"
      memory: 2Gi
  logging: ⑫
    type: inline
    loggers:
      mirrormaker.root.logger: "INFO"
  readinessProbe: ⑬
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  metricsConfig: ⑭
   type: jmxPrometheusExporter
   valueFrom:
     configMapKeyRef:
       name: my-config-map
       key: my-key
  jvmOptions: ⑮
    "-Xmx": "1g"
    "-Xms": "1g"
  image: my-org/my-image:latest ⑯
  template: ⑰
    pod:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: application
                    operator: In
                    values:
                      - postgresql
                      - mongodb
              topologyKey: "kubernetes.io/hostname"
    mirrorMakerContainer: ⑱
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing: ⑲
    type: opentelemetry
```

① The number of replica nodes.

② Bootstrap servers for consumer and producer.

③ Group ID for the consumer.

④ The number of consumer streams.

⑤ The offset auto-commit interval in milliseconds.

⑥ TLS encryption with key names under which TLS certificates are stored in X.509 format for consumer or producer. If certificates are stored in the same secret, it can be listed multiple times.

⑦ Authentication for consumer or producer, specified as mTLS, token-based OAuth, SASL-based SCRAM-SHA-256/SCRAM-SHA-512, or PLAIN.

⑧ Kafka configuration options for consumer and producer.

⑨ If the `abortOnSendFailure` property is set to `true`, Kafka MirrorMaker will exit and the container will restart following a send failure for a message.

⑩ A list of included topics mirrored from source to target Kafka cluster.

⑪ Requests for reservation of supported resources, currently `cpu` and `memory`, and limits to specify the maximum resources that can be consumed.

⑫ Specified loggers and log levels added directly (`inline`) or indirectly (`external`) through a ConfigMap. A custom ConfigMap must be placed under the `log4j.properties` or `log4j2.properties` key. MirrorMaker has a single logger called `mirrormaker.root.logger`. You can set the log level to INFO, ERROR, WARN, TRACE, DEBUG, FATAL or OFF.

⑬ Healthchecks to know when to restart a container (liveness) and when a container can accept traffic (readiness).

⑭ Prometheus metrics, which are enabled by referencing a ConfigMap containing configuration for the Prometheus JMX exporter in this example. You can enable metrics without further configuration using a reference to a ConfigMap containing an empty file under `metricsConfig.valueFrom.configMapKeyRef.key`.

⑮ JVM configuration options to optimize performance for the Virtual Machine (VM) running Kafka MirrorMaker.

⑯ ADVANCED OPTION: Container image configuration, which is recommended only in special situations.

⑰ Template customization. Here a pod is scheduled with anti-affinity, so the pod is not scheduled on nodes with the same hostname.

⑱ Environment variables are set for distributed tracing.

⑲ Distributed tracing is enabled by using OpenTelemetry.

WARNING	With the `abortOnSendFailure` property set to `false`, the producer attempts to send the next message in a topic. The original message might be lost, as there is no attempt to resend a failed message.

2. Create or update the resource:

```
kubectl apply -f <your-file>
```

Additional resources

- Introducing distributed tracing

2.4.2. List of Kafka MirrorMaker cluster resources

The following resources are created by the Cluster Operator in the Kubernetes cluster:

<*mirror-maker-name*>-mirror-maker

Deployment which is responsible for creating the Kafka MirrorMaker pods.

<*mirror-maker-name*>-config

ConfigMap which contains ancillary configuration for the Kafka MirrorMaker, and is mounted as a volume by the Kafka broker pods.

<*mirror-maker-name*>-mirror-maker

Pod Disruption Budget configured for the Kafka MirrorMaker worker nodes.

2.5. Kafka Bridge cluster configuration

Configure a Kafka Bridge deployment using the `KafkaBridge` resource. Kafka Bridge provides an API for integrating HTTP-based clients with a Kafka cluster.

`KafkaBridge` schema reference describes the full schema of the `KafkaBridge` resource.

2.5.1. Configuring the Kafka Bridge

Use the Kafka Bridge to make HTTP-based requests to the Kafka cluster.

Use the properties of the `KafkaBridge` resource to configure your Kafka Bridge deployment.

In order to prevent issues arising when client consumer requests are processed by different Kafka Bridge instances, address-based routing must be employed to ensure that requests are routed to the right Kafka Bridge instance. Additionally, each independent Kafka Bridge instance must have a replica. A Kafka Bridge instance has its own state which is not shared with another instances.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

See the *Deploying and Upgrading Strimzi* guide for instructions on running a:

- Cluster Operator
- Kafka cluster

Procedure

1. Edit the `spec` properties for the `KafkaBridge` resource.

   The properties you can configure are shown in this example configuration:

   ```
   apiVersion: kafka.strimzi.io/v1beta2
   kind: KafkaBridge
   ```

```
metadata:
  name: my-bridge
spec:
  replicas: 3 ①
  bootstrapServers: <cluster_name>-cluster-kafka-bootstrap:9092 ②
  tls: ③
    trustedCertificates:
      - secretName: my-cluster-cluster-cert
        certificate: ca.crt
      - secretName: my-cluster-cluster-cert
        certificate: ca2.crt
  authentication: ④
    type: tls
    certificateAndKey:
      secretName: my-secret
      certificate: public.crt
      key: private.key
  http: ⑤
    port: 8080
    cors: ⑥
      allowedOrigins: "https://strimzi.io"
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  consumer: ⑦
    config:
      auto.offset.reset: earliest
  producer: ⑧
    config:
      delivery.timeout.ms: 300000
  resources: ⑨
    requests:
      cpu: "1"
      memory: 2Gi
    limits:
      cpu: "2"
      memory: 2Gi
  logging: ⑩
    type: inline
    loggers:
      logger.bridge.level: "INFO"
      # enabling DEBUG just for send operation
      logger.send.name: "http.openapi.operation.send"
      logger.send.level: "DEBUG"
  jvmOptions: ⑪
    "-Xmx": "1g"
    "-Xms": "1g"
  readinessProbe: ⑫
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
```

```
  image: my-org/my-image:latest ⑬
  template: ⑭
    pod:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: application
                    operator: In
                    values:
                      - postgresql
                      - mongodb
              topologyKey: "kubernetes.io/hostname"
    bridgeContainer: ⑮
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_OTLP_ENDPOINT
          value: "http://otlp-host:4317"
  tracing:
    type: opentelemetry ⑯
```

① The number of replica nodes.

② Bootstrap server for connection to the target Kafka cluster. Use the name of the Kafka cluster as the <*cluster_name*>.

③ TLS encryption with key names under which TLS certificates are stored in X.509 format for the source Kafka cluster. If certificates are stored in the same secret, it can be listed multiple times.

④ Authentication for the Kafka Bridge cluster, specified as mTLS, token-based OAuth, SASL-based SCRAM-SHA-256/SCRAM-SHA-512, or PLAIN. By default, the Kafka Bridge connects to Kafka brokers without authentication.

⑤ HTTP access to Kafka brokers.

⑥ CORS access specifying selected resources and access methods. Additional HTTP headers in requests describe the origins that are permitted access to the Kafka cluster.

⑦ Consumer configuration options.

⑧ Producer configuration options.

⑨ Requests for reservation of supported resources, currently `cpu` and `memory`, and limits to specify the maximum resources that can be consumed.

⑩ Specified Kafka Bridge loggers and log levels added directly (`inline`) or indirectly (`external`) through a ConfigMap. A custom ConfigMap must be placed under the `log4j.properties` or `log4j2.properties` key. For the Kafka Bridge loggers, you can set the log level to INFO, ERROR, WARN, TRACE, DEBUG, FATAL or OFF.

⑪ JVM configuration options to optimize performance for the Virtual Machine (VM) running the Kafka Bridge.

⑫ Healthchecks to know when to restart a container (liveness) and when a container can accept traffic (readiness).

⑬ Optional: Container image configuration, which is recommended only in special situations.

⑭ Template customization. Here a pod is scheduled with anti-affinity, so the pod is not scheduled on nodes with the same hostname.

⑮ Environment variables are set for distributed tracing.

⑯ Distributed tracing is enabled by using OpenTelemetry.

2. Create or update the resource:

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

Additional resources

- Using the Strimzi Kafka Bridge
- Introducing distributed tracing

2.5.2. List of Kafka Bridge cluster resources

The following resources are created by the Cluster Operator in the Kubernetes cluster:

***bridge-cluster-name*-bridge**

Deployment which is in charge to create the Kafka Bridge worker node pods.

***bridge-cluster-name*-bridge-service**

Service which exposes the REST interface of the Kafka Bridge cluster.

***bridge-cluster-name*-bridge-config**

ConfigMap which contains the Kafka Bridge ancillary configuration and is mounted as a volume by the Kafka broker pods.

***bridge-cluster-name*-bridge**

Pod Disruption Budget configured for the Kafka Bridge worker nodes.

2.6. Customizing Kubernetes resources

A Strimzi deployment creates Kubernetes resources, such as `Deployment`, `Pod`, and `Service` resources. These resources are managed by Strimzi operators. Only the operator that is responsible for managing a particular Kubernetes resource can change that resource. If you try to manually change an operator-managed Kubernetes resource, the operator will revert your changes back.

Changing an operator-managed Kubernetes resource can be useful if you want to perform certain tasks, such as:

- Adding custom labels or annotations that control how `Pods` are treated by Istio or other services
- Managing how `Loadbalancer`-type Services are created by the cluster

You can make the changes using the `template` property in the Strimzi custom resources. The `template` property is supported in the following resources. The API reference provides more details about the customizable fields.

`Kafka.spec.kafka`

See `KafkaClusterTemplate` schema reference

`Kafka.spec.zookeeper`

See `ZookeeperClusterTemplate` schema reference

`Kafka.spec.entityOperator`

See `EntityOperatorTemplate` schema reference

`Kafka.spec.kafkaExporter`

See `KafkaExporterTemplate` schema reference

`Kafka.spec.cruiseControl`

See `CruiseControlTemplate` schema reference

`KafkaConnect.spec`

See `KafkaConnectTemplate` schema reference

`KafkaMirrorMaker.spec`

See `KafkaMirrorMakerTemplate` schema reference

`KafkaMirrorMaker2.spec`

See `KafkaConnectTemplate` schema reference

`KafkaBridge.spec`

See `KafkaBridgeTemplate` schema reference

`KafkaUser.spec`

See `KafkaUserTemplate` schema reference

In the following example, the `template` property is used to modify the labels in a Kafka broker's pod.

Example template customization

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

```
        metadata:
          labels:
            mylabel: myvalue
    # ...
```

2.6.1. Customizing the image pull policy

Strimzi allows you to customize the image pull policy for containers in all pods deployed by the Cluster Operator. The image pull policy is configured using the environment variable `STRIMZI_IMAGE_PULL_POLICY` in the Cluster Operator deployment. The `STRIMZI_IMAGE_PULL_POLICY` environment variable can be set to three different values:

`Always`
: Container images are pulled from the registry every time the pod is started or restarted.

`IfNotPresent`
: Container images are pulled from the registry only when they were not pulled before.

`Never`
: Container images are never pulled from the registry.

Currently, the image pull policy can only be customized for all Kafka, Kafka Connect, and Kafka MirrorMaker clusters at once. Changing the policy will result in a rolling update of all your Kafka, Kafka Connect, and Kafka MirrorMaker clusters.

Additional resources

- Using the Cluster Operator.
- Disruptions.

2.6.2. Applying a termination grace period

Apply a termination grace period to give a Kafka cluster enough time to shut down cleanly.

Specify the time using the `terminationGracePeriodSeconds` property. Add the property to the `template.pod` configuration of the `Kafka` custom resource.

The time you add will depend on the size of your Kafka cluster. The Kubernetes default for the termination grace period is 30 seconds. If you observe that your clusters are not shutting down cleanly, you can increase the termination grace period.

A termination grace period is applied every time a pod is restarted. The period begins when Kubernetes sends a *term* (termination) signal to the processes running in the pod. The period should reflect the amount of time required to transfer the processes of the terminating pod to another pod before they are stopped. After the period ends, a *kill* signal stops any processes still running in the pod.

The following example adds a termination grace period of 120 seconds to the `Kafka` custom resource. You can also specify the configuration in the custom resources of other Kafka

components.

Example termination grace period configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    template:
      pod:
        terminationGracePeriodSeconds: 120
        # ...
    # ...
```

2.7. Configuring pod scheduling

When two applications are scheduled to the same Kubernetes node, both applications might use the same resources like disk I/O and impact performance. That can lead to performance degradation. Scheduling Kafka pods in a way that avoids sharing nodes with other critical workloads, using the right nodes or dedicated a set of nodes only for Kafka are the best ways how to avoid such problems.

2.7.1. Specifying affinity, tolerations, and topology spread constraints

Use affinity, tolerations and topology spread constraints to schedule the pods of kafka resources onto nodes. Affinity, tolerations and topology spread constraints are configured using the `affinity`, `tolerations`, and `topologySpreadConstraint` properties in following resources:

- `Kafka.spec.kafka.template.pod`
- `Kafka.spec.zookeeper.template.pod`
- `Kafka.spec.entityOperator.template.pod`
- `KafkaConnect.spec.template.pod`
- `KafkaBridge.spec.template.pod`
- `KafkaMirrorMaker.spec.template.pod`
- `KafkaMirrorMaker2.spec.template.pod`

The format of the `affinity`, `tolerations`, and `topologySpreadConstraint` properties follows the Kubernetes specification. The affinity configuration can include different types of affinity:

- Pod affinity and anti-affinity
- Node affinity

Additional resources

- Kubernetes node and pod affinity documentation
- Kubernetes taints and tolerations
- Kubernetes Topology Spread Constraints

Use pod anti-affinity to avoid critical applications sharing nodes

Use pod anti-affinity to ensure that critical applications are never scheduled on the same disk. When running a Kafka cluster, it is recommended to use pod anti-affinity to ensure that the Kafka brokers do not share nodes with other workloads, such as databases.

Use node affinity to schedule workloads onto specific nodes

The Kubernetes cluster usually consists of many different types of worker nodes. Some are optimized for CPU heavy workloads, some for memory, while other might be optimized for storage (fast local SSDs) or network. Using different nodes helps to optimize both costs and performance. To achieve the best possible performance, it is important to allow scheduling of Strimzi components to use the right nodes.

Kubernetes uses node affinity to schedule workloads onto specific nodes. Node affinity allows you to create a scheduling constraint for the node on which the pod will be scheduled. The constraint is specified as a label selector. You can specify the label using either the built-in node label like `beta.kubernetes.io/instance-type` or custom labels to select the right node.

Use node affinity and tolerations for dedicated nodes

Use taints to create dedicated nodes, then schedule Kafka pods on the dedicated nodes by configuring node affinity and tolerations.

Cluster administrators can mark selected Kubernetes nodes as tainted. Nodes with taints are excluded from regular scheduling and normal pods will not be scheduled to run on them. Only services which can tolerate the taint set on the node can be scheduled on it. The only other services running on such nodes will be system services such as log collectors or software defined networks.

Running Kafka and its components on dedicated nodes can have many advantages. There will be no other applications running on the same nodes which could cause disturbance or consume the resources needed for Kafka. That can lead to improved performance and stability.

2.7.2. Configuring pod anti-affinity to schedule each Kafka broker on a different worker node

Many Kafka brokers or ZooKeeper nodes can run on the same Kubernetes worker node. If the worker node fails, they will all become unavailable at the same time. To improve reliability, you can use `podAntiAffinity` configuration to schedule each Kafka broker or ZooKeeper node on a different Kubernetes worker node.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

Procedure

1. Edit the `affinity` property in the resource specifying the cluster deployment. To make sure that no worker nodes are shared by Kafka brokers or ZooKeeper nodes, use the `strimzi.io/name` label. Set the `topologyKey` to `kubernetes.io/hostname` to specify that the selected pods are not scheduled on nodes with the same hostname. This will still allow the same worker node to be shared by a single Kafka broker and a single ZooKeeper node. For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: strimzi.io/name
                      operator: In
                      values:
                        - CLUSTER-NAME-kafka
                topologyKey: "kubernetes.io/hostname"
    # ...
  zookeeper:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: strimzi.io/name
                      operator: In
                      values:
                        - CLUSTER-NAME-zookeeper
                topologyKey: "kubernetes.io/hostname"
    # ...
```

Where `CLUSTER-NAME` is the name of your Kafka custom resource.

2. If you even want to make sure that a Kafka broker and ZooKeeper node do not share the same worker node, use the `strimzi.io/cluster` label. For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
```

```
kafka:
  # ...
  template:
    pod:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: strimzi.io/cluster
                    operator: In
                    values:
                      - CLUSTER-NAME
              topologyKey: "kubernetes.io/hostname"
  # ...
zookeeper:
  # ...
  template:
    pod:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: strimzi.io/cluster
                    operator: In
                    values:
                      - CLUSTER-NAME
              topologyKey: "kubernetes.io/hostname"
  # ...
```

Where `CLUSTER-NAME` is the name of your Kafka custom resource.

3. Create or update the resource.

```
kubectl apply -f <kafka_configuration_file>
```

2.7.3. Configuring pod anti-affinity in Kafka components

Pod anti-affinity configuration helps with the stability and performance of Kafka brokers. By using `podAntiAffinity`, Kubernetes will not schedule Kafka brokers on the same nodes as other workloads. Typically, you want to avoid Kafka running on the same worker node as other network or storage intensive applications such as databases, storage or other messaging platforms.

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

Procedure

1. Edit the `affinity` property in the resource specifying the cluster deployment. Use labels to specify the pods which should not be scheduled on the same nodes. The `topologyKey` should be set to `kubernetes.io/hostname` to specify that the selected pods should not be scheduled on nodes with the same hostname. For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: application
                      operator: In
                      values:
                        - postgresql
                        - mongodb
                topologyKey: "kubernetes.io/hostname"
    # ...
  zookeeper:
    # ...
```

2. Create or update the resource.

   This can be done using `kubectl apply`:

```
kubectl apply -f <kafka_configuration_file>
```

2.7.4. Configuring node affinity in Kafka components

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

Procedure

1. Label the nodes where Strimzi components should be scheduled.

   This can be done using `kubectl label`:

```
kubectl label node NAME-OF-NODE node-type=fast-network
```

Alternatively, some of the existing labels might be reused.

2. Edit the `affinity` property in the resource specifying the cluster deployment. For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        affinity:
          nodeAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              nodeSelectorTerms:
                - matchExpressions:
                  - key: node-type
                    operator: In
                    values:
                    - fast-network
    # ...
  zookeeper:
    # ...
```

3. Create or update the resource.

   This can be done using `kubectl apply`:

```
kubectl apply -f <kafka_configuration_file>
```

2.7.5. Setting up dedicated nodes and scheduling pods on them

Prerequisites

- A Kubernetes cluster
- A running Cluster Operator

Procedure

1. Select the nodes which should be used as dedicated.
2. Make sure there are no workloads scheduled on these nodes.
3. Set the taints on the selected nodes:

   This can be done using `kubectl taint`:

```
kubectl taint node NAME-OF-NODE dedicated=Kafka:NoSchedule
```

4. Additionally, add a label to the selected nodes as well.

This can be done using `kubectl label`:

```
kubectl label node NAME-OF-NODE dedicated=Kafka
```

5. Edit the `affinity` and `tolerations` properties in the resource specifying the cluster deployment.

   For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    template:
      pod:
        tolerations:
          - key: "dedicated"
            operator: "Equal"
            value: "Kafka"
            effect: "NoSchedule"
        affinity:
          nodeAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              nodeSelectorTerms:
              - matchExpressions:
                - key: dedicated
                  operator: In
                  values:
                  - Kafka
    # ...
  zookeeper:
    # ...
```

6. Create or update the resource.

   This can be done using `kubectl apply`:

```
kubectl apply -f <kafka_configuration_file>
```

2.8. Logging configuration

Configure logging levels in the custom resources of Kafka components and Strimzi Operators. You can specify the logging levels directly in the `spec.logging` property of the custom resource. Or you can define the logging properties in a ConfigMap that's referenced in the custom resource using the `configMapKeyRef` property.

The advantages of using a ConfigMap are that the logging properties are maintained in one place

and are accessible to more than one resource. You can also reuse the ConfigMap for more than one resource. If you are using a ConfigMap to specify loggers for Strimzi Operators, you can also append the logging specification to add filters.

You specify a logging `type` in your logging specification:

- `inline` when specifying logging levels directly
- `external` when referencing a ConfigMap

Example `inline` logging configuration

```
spec:
  # ...
  logging:
    type: inline
    loggers:
      kafka.root.logger.level: "INFO"
```

Example `external` logging configuration

```
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: my-config-map
        key: my-config-map-key
```

Values for the `name` and `key` of the ConfigMap are mandatory. Default logging is used if the `name` or `key` is not set.

2.8.1. Logging options for Kafka components and operators

For more information on configuring logging for specific Kafka components or operators, see the following sections.

Kafka component logging

- Kafka logging
- ZooKeeper logging
- Kafka Connect and Mirror Maker 2.0 logging
- MirrorMaker logging
- Kafka Bridge logging
- Cruise Control logging

Operator logging

- Cluster Operator logging
- Topic Operator logging
- User Operator logging

2.8.2. Creating a ConfigMap for logging

To use a ConfigMap to define logging properties, you create the ConfigMap and then reference it as part of the logging definition in the `spec` of a resource.

The ConfigMap must contain the appropriate logging configuration.

- `log4j.properties` for Kafka components, ZooKeeper, and the Kafka Bridge
- `log4j2.properties` for the Topic Operator and User Operator

The configuration must be placed under these properties.

In this procedure a ConfigMap defines a root logger for a Kafka resource.

Procedure

1. Create the ConfigMap.

   You can create the ConfigMap as a YAML file or from a properties file.

   ConfigMap example with a root logger definition for Kafka:

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: logging-configmap
   data:
     log4j.properties:
       kafka.root.logger.level="INFO"
   ```

   If you are using a properties file, specify the file at the command line:

   ```
   kubectl create configmap logging-configmap --from-file=log4j.properties
   ```

   The properties file defines the logging configuration:

   ```
   # Define the logger
   kafka.root.logger.level="INFO"
   # ...
   ```

2. Define *external* logging in the `spec` of the resource, setting the `logging.valueFrom.configMapKeyRef.name` to the name of the ConfigMap and `logging.valueFrom.configMapKeyRef.key` to the key in this ConfigMap.

```
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: logging-configmap
        key: log4j.properties
```

3. Create or update the resource.

```
kubectl apply -f <kafka_configuration_file>
```

2.8.3. Adding logging filters to Operators

If you are using a ConfigMap to configure the (log4j2) logging levels for Strimzi Operators, you can also define logging filters to limit what's returned in the log.

Logging filters are useful when you have a large number of logging messages. Suppose you set the log level for the logger as DEBUG (`rootLogger.level="DEBUG"`). Logging filters reduce the number of logs returned for the logger at that level, so you can focus on a specific resource. When the filter is set, only log messages matching the filter are logged.

Filters use *markers* to specify what to include in the log. You specify a kind, namespace and name for the marker. For example, if a Kafka cluster is failing, you can isolate the logs by specifying the kind as `Kafka`, and use the namespace and name of the failing cluster.

This example shows a marker filter for a Kafka cluster named `my-kafka-cluster`.

Basic logging filter configuration

```
rootLogger.level="INFO"
appender.console.filter.filter1.type=MarkerFilter ①
appender.console.filter.filter1.onMatch=ACCEPT ②
appender.console.filter.filter1.onMismatch=DENY ③
appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster) ④
```

① The `MarkerFilter` type compares a specified marker for filtering.

② The `onMatch` property accepts the log if the marker matches.

③ The `onMismatch` property rejects the log if the marker does not match.

④ The marker used for filtering is in the format *KIND(NAMESPACE/NAME-OF-RESOURCE)*.

You can create one or more filters. Here, the log is filtered for two Kafka clusters.

Multiple logging filter configuration

```
appender.console.filter.filter1.type=MarkerFilter
```

```
appender.console.filter.filter1.onMatch=ACCEPT
appender.console.filter.filter1.onMismatch=DENY
appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster-1)
appender.console.filter.filter2.type=MarkerFilter
appender.console.filter.filter2.onMatch=ACCEPT
appender.console.filter.filter2.onMismatch=DENY
appender.console.filter.filter2.marker=Kafka(my-namespace/my-kafka-cluster-2)
```

Adding filters to the Cluster Operator

To add filters to the Cluster Operator, update its logging ConfigMap YAML file (`install/cluster-operator/050-ConfigMap-strimzi-cluster-operator.yaml`).

Procedure

1. Update the `050-ConfigMap-strimzi-cluster-operator.yaml` file to add the filter properties to the ConfigMap.

   In this example, the filter properties return logs only for the `my-kafka-cluster` Kafka cluster:

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: strimzi-cluster-operator
   data:
     log4j2.properties:
       #...
       appender.console.filter.filter1.type=MarkerFilter
       appender.console.filter.filter1.onMatch=ACCEPT
       appender.console.filter.filter1.onMismatch=DENY
       appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster)
   ```

   Alternatively, edit the `ConfigMap` directly:

   ```
   kubectl edit configmap strimzi-cluster-operator
   ```

2. If you updated the YAML file instead of editing the `ConfigMap` directly, apply the changes by deploying the ConfigMap:

   ```
   kubectl create -f install/cluster-operator/050-ConfigMap-strimzi-cluster-
   operator.yaml
   ```

Adding filters to the Topic Operator or User Operator

To add filters to the Topic Operator or User Operator, create or edit a logging ConfigMap.

In this procedure a logging ConfigMap is created with filters for the Topic Operator. The same approach is used for the User Operator.

Procedure

1. Create the ConfigMap.

   You can create the ConfigMap as a YAML file or from a properties file.

   In this example, the filter properties return logs only for the `my-topic` topic:

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: logging-configmap
   data:
     log4j2.properties:
       rootLogger.level="INFO"
       appender.console.filter.filter1.type=MarkerFilter
       appender.console.filter.filter1.onMatch=ACCEPT
       appender.console.filter.filter1.onMismatch=DENY
       appender.console.filter.filter1.marker=KafkaTopic(my-namespace/my-topic)
   ```

   If you are using a properties file, specify the file at the command line:

   ```
   kubectl create configmap logging-configmap --from-file=log4j2.properties
   ```

   The properties file defines the logging configuration:

   ```
   # Define the logger
   rootLogger.level="INFO"
   # Set the filters
   appender.console.filter.filter1.type=MarkerFilter
   appender.console.filter.filter1.onMatch=ACCEPT
   appender.console.filter.filter1.onMismatch=DENY
   appender.console.filter.filter1.marker=KafkaTopic(my-namespace/my-topic)
   # ...
   ```

2. Define *external* logging in the `spec` of the resource, setting the `logging.valueFrom.configMapKeyRef.name` to the name of the ConfigMap and `logging.valueFrom.configMapKeyRef.key` to the key in this ConfigMap.

   For the Topic Operator, logging is specified in the `topicOperator` configuration of the `Kafka` resource.

   ```
   spec:
     # ...
     entityOperator:
       topicOperator:
         logging:
           type: external
   ```

```
        valueFrom:
          configMapKeyRef:
            name: logging-configmap
            key: log4j2.properties
```

3. Apply the changes by deploying the Cluster Operator:

```
create -f install/cluster-operator -n my-cluster-operator-namespace
```

Additional resources

- Configuring Kafka
- Cluster Operator logging
- Topic Operator logging
- User Operator logging

Chapter 3. Applying security context to Strimzi pods and containers

Security context defines constraints on pods and containers. By specifying a security context, pods and containers only have the permissions they need. For example, permissions can control runtime operations or access to resources.

3.1. How to configure security context

Use security provider plugins or template configuration to apply security context to Strimzi pods and containers.

Apply security context at the pod or container level:

Pod-level security context

Pod-level security context is applied to all containers in a specific pod.

Container-level security context

Container-level security context is applied to a specific container.

With Strimzi, security context is applied through one or both of the following methods:

Template configuration

Use `template` configuration of Strimzi custom resources to specify security context at the pod or container level.

Pod security provider plugins

Use pod security provider plugins to automatically set security context across all pods and containers using preconfigured settings.

Pod security providers offer a simpler alternative to specifying security context through `template` configuration. You can use both approaches. The `template` approach has a higher priority. Security context configured through `template` properties overrides the configuration set by pod security providers. So you might use pod security providers to automatically configure the security context for most containers. And also use `template` configuration to set container-specific security context where needed.

The `template` approach provides flexibility, but it also means you have to configure security context in numerous places to capture the security you want for all pods and containers. For example, you'll need to apply the configuration to each pod in a Kafka cluster, as well as the pods for deployments of other Kafka components.

To avoid repeating the same configuration, you can use the following pod security provider plugins so that the security configuration is in one place.

Baseline Provider

The Baseline Provider is based on the Kubernetes *baseline* security profile. The baseline profile

prevents privilege escalations and defines other standard access controls and limitations.

Restricted Provider

The Restricted Provider is based on the Kubernetes *restricted* security profile. The restricted profile is more restrictive than the baseline profile, and is used where security needs to be tighter.

For more information on the Kubernetes security profiles, see Pod security standards.

3.1.1. Template configuration for security context

In the following example, security context is configured for Kafka brokers in the `template` configuration of the `Kafka` resource. Security context is specified at the pod and container level.

Example `template` configuration for security context

```
apiVersion: {KafkaApiVersion}
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  kafka:
    template:
      pod: ①
        securityContext:
          runAsUser: 1000001
          fsGroup: 0
      kafkaContainer: ②
        securityContext:
          runAsUser: 2000
  # ...
```

① Pod security context

② Container security context of the Kafka broker container

3.1.2. Baseline Provider for pod security

The Baseline Provider is the default pod security provider. It configures the pods managed by Strimzi with a baseline security profile. The baseline profile is compatible with previous versions of Strimzi.

The Baseline Provider is enabled by default if you don't specify a provider. Though you can enable it explicitly by setting the `STRIMZI_POD_SECURITY_PROVIDER_CLASS` environment variable to `baseline` when configuring the Cluster Operator.

Configuration for the Baseline Provider

```
# ...
env:
```

```
  # ...
  - name: STRIMZI_POD_SECURITY_PROVIDER_CLASS
    value: baseline
  # ...
```

Instead of specifying `baseline` as the value, you can specify the `io.strimzi.plugin.security.profiles.impl.BaselinePodSecurityProvider` fully-qualified domain name.

3.1.3. Restricted Provider for pod security

The Restricted Provider provides a higher level of security than the Baseline Provider. It configures the pods managed by Strimzi with a restricted security profile.

You enable the Restricted Provider by setting the `STRIMZI_POD_SECURITY_PROVIDER_CLASS` environment variable to `restricted` when configuring the Cluster Operator.

Configuration for the Restricted Provider

```
# ...
env:
  # ...
  - name: STRIMZI_POD_SECURITY_PROVIDER_CLASS
    value: restricted
  # ...
```

Instead of specifying `restricted` as the value, you can specify the `io.strimzi.plugin.security.profiles.impl.RestrictedPodSecurityProvider` fully-qualified domain name.

If you change to the Restricted Provider from the default Baseline Provider, the following restrictions are implemented in addition to the constraints defined in the baseline security profile:

- Limits allowed volume types
- Disallows privilege escalation
- Requires applications to run under a non-root user
- Requires `seccomp` (secure computing mode) profiles to be set as `RuntimeDefault` or `Localhost`
- Limits container capabilities to use only the `NET_BIND_SERVICE` capability

With the Restricted Provider enabled, containers created by the Cluster Operator are set with the following security context.

Cluster Operator with restricted security context configuration

```
# ...
securityContext:
  allowPrivilegeEscalation: false
  capabilities:
```

```
    drop:
      - ALL
  runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
# ...
```

NOTE

Container capabilities and `seccomp` are Linux kernel features that support container security.

- Capabilities add fine-grained privileges for processes running on a container. The `NET_BIND_SERVICE` capability allows non-root user applications to bind to ports below 1024.
- `seccomp` profiles limit the processes running in a container to only a subset of system calls. The `RuntimeDefault` profile provides a default set of system calls. A `LocalHost` profile uses a profile defined in a file on the node.

Additional resources

- Security context on Kubernetes
- Pod security standards on Kubernetes (including profile descriptions)
- `ContainerTemplate` schema reference

3.2. Enabling the Restricted Provider for the Cluster Operator

Security pod providers configure the security context constraints of the pods and containers created by the Cluster Operator. The Baseline Provider is the default pod security provider used by Strimzi. You can switch to the Restricted Provider by changing the `STRIMZI_POD_SECURITY_PROVIDER_CLASS` environment variable in the Cluster Operator configuration.

To make the required changes, configure the `060-Deployment-strimzi-cluster-operator.yaml` Cluster Operator installation file located in `install/cluster-operator/`.

By enabling a new pod security provider, any pods or containers created by the Cluster Operator are subject to the limitations it imposes. Pods and containers that are already running are restarted for the changes to take affect.

Prerequisites

- You need an account with permission to create and manage `CustomResourceDefinition` and RBAC (`ClusterRole`, and `RoleBinding`) resources.

Procedure

Edit the `Deployment` resource that is used to deploy the Cluster Operator, which is defined in the `060-Deployment-strimzi-cluster-operator.yaml` file.

1. Add or amend the `STRIMZI_POD_SECURITY_PROVIDER_CLASS` environment variable with a value of

`restricted`.

Cluster Operator configuration for the Restricted Provider

```
# ...
env:
  # ...
  - name: STRIMZI_POD_SECURITY_PROVIDER_CLASS
    value: restricted
  # ...
```

Or you can specify the `io.strimzi.plugin.security.profiles.impl.RestrictedPodSecurityProvider` fully-qualified domain name.

2. Deploy the Cluster Operator:

```
kubectl create -f install/cluster-operator -n myproject
```

3. (Optional) Use `template` configuration to set security context for specific components at the pod or container level.

Adding security context through `template` configuration

```
template:
  pod:
    securityContext:
      runAsUser: 1000001
      fsGroup: 0
  kafkaContainer:
    securityContext:
    runAsUser: 2000
  # ...
```

If you apply specific security context for a component using `template` configuration, it takes priority over the general configuration provided by the pod security provider.

3.3. Implementing a custom pod security provider

If Strimzi's Baseline Provider and Restricted Provider don't quite match your needs, you can develop a custom pod security provider to deliver all-encompassing pod and container security context constraints.

Implement a custom pod security provider to apply your own security context profile. You can decide what applications and privileges to include in the profile.

Your custom pod security provider can implement the `PodSecurityProvider.java` interface that gets the security context for pods and containers; or it can extend the Baseline Provider or Restricted

Provider classes.

The pod security provider plugins use the Java Service Provider Interface, so your custom pod security provider also requires a provider configuration file for service discovery.

To implement your own provider, the general steps include the following:

1. Build the JAR file for the provider.
2. Add the JAR file to the Cluster Operator image.
3. Specify the custom pod security provider when setting the Cluster Operator environment variable `STRIMZI_POD_SECURITY_PROVIDER_CLASS`.

Additional resources

- Pod security provider interface
- Baseline Provider and Restricted Provider classes
- Provider configuration file
- Java Service Provider Interface

3.4. Handling of security context by Kubernetes platform

Handling of security context depends on the tooling of the Kubernetes platform you are using.

For example, OpenShift uses built-in security context constraints (SCCs) to control permissions. SCCs are the settings and strategies that control the security features a pod has access to.

By default, OpenShift injects security context configuration automatically. In most cases, this means you don't need to configure security context for the pods and containers created by the Cluster Operator. Although you can still create and manage your own SCCs.

For more information, see the OpenShift documentation.

Chapter 4. Custom resource API reference

4.1. Common configuration properties

Use Common configuration properties to configure Strimzi custom resources. You add common configuration properties to a custom resource like any other supported configuration for that resource.

4.1.1. `replicas`

Use the `replicas` property to configure replicas.

The type of replication depends on the resource.

- `KafkaTopic` uses a replication factor to configure the number of replicas of each partition within a Kafka cluster.
- Kafka components use replicas to configure the number of pods in a deployment to provide better availability and scalability.

NOTE	When running a Kafka component on Kubernetes it may not be necessary to run multiple replicas for high availability. When the node where the component is deployed crashes, Kubernetes will automatically reschedule the Kafka component pod to a different node. However, running Kafka components with multiple replicas can provide faster failover times as the other nodes will be up and running.

4.1.2. `bootstrapServers`

Use the `bootstrapServers` property to configure a list of bootstrap servers.

The bootstrap server lists can refer to Kafka clusters that are not deployed in the same Kubernetes cluster. They can also refer to a Kafka cluster not deployed by Strimzi.

If on the same Kubernetes cluster, each list must ideally contain the Kafka cluster bootstrap service which is named `CLUSTER-NAME-kafka-bootstrap` and a port number. If deployed by Strimzi but on different Kubernetes clusters, the list content depends on the approach used for exposing the clusters (routes, ingress, nodeports or loadbalancers).

When using Kafka with a Kafka cluster not managed by Strimzi, you can specify the bootstrap servers list according to the configuration of the given cluster.

4.1.3. `ssl` (supported TLS versions and cipher suites)

You can incorporate SSL configuration and cipher suite specifications to further secure TLS-based communication between your client application and a Kafka cluster. In addition to the standard TLS configuration, you can specify a supported TLS version and enable cipher suites in the configuration for the Kafka broker. You can also add the configuration to your clients if you wish to limit the TLS versions and cipher suites they use. The configuration on the client must only use

protocols and cipher suites that are enabled on the broker.

A cipher suite is a set of security mechanisms for secure connection and data transfer. For example, the cipher suite `TLS_AES_256_GCM_SHA384` is composed of the following mechanisms, which are used in conjunction with the TLS protocol:

- AES (Advanced Encryption Standard) encryption (256-bit key)
- GCM (Galois/Counter Mode) authenticated encryption
- SHA384 (Secure Hash Algorithm) data integrity protection

The combination is encapsulated in the `TLS_AES_256_GCM_SHA384` cipher suite specification.

The `ssl.enabled.protocols` property specifies the available TLS versions that can be used for secure communication between the cluster and its clients. The `ssl.protocol` property sets the default TLS version for all connections, and it must be chosen from the enabled protocols. Use the `ssl.endpoint.identification.algorithm` property to enable or disable hostname verification.

Example SSL configuration

```
# ...
config:
  ssl.cipher.suites: TLS_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ①
  ssl.enabled.protocols: TLSv1.3, TLSv1.2 ②
  ssl.protocol: TLSv1.3 ③
  ssl.endpoint.identification.algorithm: HTTPS ④
# ...
```

① Cipher suite specifications enabled.

② TLS versions supported.

③ Default TLS version is `TLSv1.3`. If a client only supports TLSv1.2, it can still connect to the broker and communicate using that supported version, and vice versa if the configuration is on the client and the broker only supports TLSv1.2.

④ Hostname verification is enabled by setting to `HTTPS`. An empty string disables the verification.

4.1.4. `trustedCertificates`

Having set `tls` to configure TLS encryption, use the `trustedCertificates` property to provide a list of secrets with key names under which the certificates are stored in X.509 format.

You can use the secrets created by the Cluster Operator for the Kafka cluster, or you can create your own TLS certificate file, then create a `Secret` from the file:

```
kubectl create secret generic MY-SECRET \
--from-file=MY-TLS-CERTIFICATE-FILE.crt
```

Example TLS encryption configuration

```
tls:
  trustedCertificates:
    - secretName: my-cluster-cluster-cert
      certificate: ca.crt
    - secretName: my-cluster-cluster-cert
      certificate: ca2.crt
```

If certificates are stored in the same secret, it can be listed multiple times.

If you want to enable TLS encryption, but use the default set of public certification authorities shipped with Java, you can specify `trustedCertificates` as an empty array:

Example of enabling TLS with the default Java certificates

```
tls:
  trustedCertificates: []
```

For information on configuring mTLS authentication, see the `KafkaClientAuthenticationTls` schema reference.

4.1.5. `resources`

Configure resource *requests* and *limits* to control resources for Strimzi containers. You can specify requests and limits for `memory` and `cpu` resources. The requests should be enough to ensure a stable performance of Kafka.

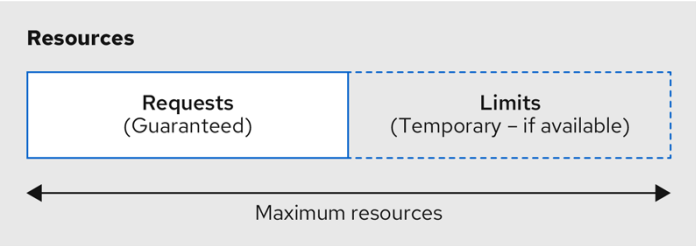
How you configure resources in a production environment depends on a number of factors. For example, applications are likely to be sharing resources in your Kubernetes cluster.

For Kafka, the following aspects of a deployment can impact the resources you need:

- Throughput and size of messages
- The number of network threads handling messages
- The number of producers and consumers
- The number of topics and partitions

The values specified for resource requests are reserved and always available to the container. Resource limits specify the maximum resources that can be consumed by a given container. The amount between the request and limit is not reserved and might not be always available. A container can use the resources up to the limit only when they are available. Resource limits are temporary and can be reallocated.

Resource requests and limits



212_Streams_0322

If you set limits without requests or vice versa, Kubernetes uses the same value for both. Setting equal requests and limits for resources guarantees quality of service, as Kubernetes will not kill containers unless they exceed their limits.

You can configure resource requests and limits for one or more supported resources.

Example resource configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    #...
    resources:
      requests:
        memory: 64Gi
        cpu: "8"
      limits:
        memory: 64Gi
        cpu: "12"
  entityOperator:
    #...
    topicOperator:
      #...
      resources:
        requests:
          memory: 512Mi
          cpu: "1"
        limits:
          memory: 512Mi
          cpu: "1"
```

Resource requests and limits for the Topic Operator and User Operator are set in the `Kafka` resource.

If the resource request is for more than the available free resources in the Kubernetes cluster, the pod is not scheduled.

NOTE Strimzi uses the Kubernetes syntax for specifying `memory` and `cpu` resources. For more information about managing computing resources on Kubernetes, see

> Managing Compute Resources for Containers.

Memory resources

When configuring memory resources, consider the total requirements of the components.

Kafka runs inside a JVM and uses an operating system page cache to store message data before writing to disk. The memory request for Kafka should fit the JVM heap and page cache. You can configure the `jvmOptions` property to control the minimum and maximum heap size.

Other components don't rely on the page cache. You can configure memory resources without configuring the `jvmOptions` to control the heap size.

Memory requests and limits are specified in megabytes, gigabytes, mebibytes, and gibibytes. Use the following suffixes in the specification:

- `M` for megabytes
- `G` for gigabytes
- `Mi` for mebibytes
- `Gi` for gibibytes

Example resources using different memory units

```
# ...
resources:
  requests:
    memory: 512Mi
  limits:
    memory: 2Gi
# ...
```

For more details about memory specification and additional supported units, see Meaning of memory.

CPU resources

A CPU request should be enough to give a reliable performance at any time. CPU requests and limits are specified as *cores* or *millicpus*/*millicores*.

CPU cores are specified as integers (`5` CPU core) or decimals (`2.5` CPU core). 1000 *millicores* is the same as `1` CPU core.

Example CPU units

```
# ...
resources:
  requests:
    cpu: 500m
  limits:
    cpu: 2.5
```

```
# ...
```

The computing power of 1 CPU core may differ depending on the platform where Kubernetes is deployed.

For more information on CPU specification, see Meaning of CPU.

4.1.6. `image`

Use the `image` property to configure the container image used by the component.

Overriding container images is recommended only in special situations where you need to use a different container registry or a customized image.

For example, if your network does not allow access to the container repository used by Strimzi, you can copy the Strimzi images or build them from the source. However, if the configured image is not compatible with Strimzi images, it might not work properly.

A copy of the container image might also be customized and used for debugging.

You can specify which container image to use for a component using the `image` property in the following resources:

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.entityOperator.tlsSidecar`
- `KafkaConnect.spec`
- `KafkaMirrorMaker.spec`
- `KafkaMirrorMaker2.spec`
- `KafkaBridge.spec`

Configuring the `image` property for Kafka, Kafka Connect, and Kafka MirrorMaker

Kafka, Kafka Connect, and Kafka MirrorMaker support multiple versions of Kafka. Each component requires its own image. The default images for the different Kafka versions are configured in the following environment variables:

- `STRIMZI_KAFKA_IMAGES`
- `STRIMZI_KAFKA_CONNECT_IMAGES`
- `STRIMZI_KAFKA_MIRROR_MAKER_IMAGES`

These environment variables contain mappings between the Kafka versions and their corresponding images. The mappings are used together with the `image` and `version` properties:

- If neither `image` nor `version` are given in the custom resource then the `version` will default to the Cluster Operator's default Kafka version, and the image will be the one corresponding to this version in the environment variable.
- If `image` is given but `version` is not, then the given image is used and the `version` is assumed to be the Cluster Operator's default Kafka version.
- If `version` is given but `image` is not, then the image that corresponds to the given version in the environment variable is used.
- If both `version` and `image` are given, then the given image is used. The image is assumed to contain a Kafka image with the given version.

The `image` and `version` for the different components can be configured in the following properties:

- For Kafka in `spec.kafka.image` and `spec.kafka.version`.
- For Kafka Connect and Kafka MirrorMaker in `spec.image` and `spec.version`.

WARNING	It is recommended to provide only the `version` and leave the `image` property unspecified. This reduces the chance of making a mistake when configuring the custom resource. If you need to change the images used for different versions of Kafka, it is preferable to configure the Cluster Operator's environment variables.

Configuring the `image` property in other resources

For the `image` property in the other custom resources, the given value will be used during deployment. If the `image` property is missing, the `image` specified in the Cluster Operator configuration will be used. If the `image` name is not defined in the Cluster Operator configuration, then the default value will be used.

- For Topic Operator:
  1. Container image specified in the `STRIMZI_DEFAULT_TOPIC_OPERATOR_IMAGE` environment variable from the Cluster Operator configuration.
  2. `quay.io/strimzi/operator:0.35.1` container image.
- For User Operator:
  1. Container image specified in the `STRIMZI_DEFAULT_USER_OPERATOR_IMAGE` environment variable from the Cluster Operator configuration.
  2. `quay.io/strimzi/operator:0.35.1` container image.
- For Entity Operator TLS sidecar:
  1. Container image specified in the `STRIMZI_DEFAULT_TLS_SIDECAR_ENTITY_OPERATOR_IMAGE` environment variable from the Cluster Operator configuration.
  2. `quay.io/strimzi/kafka:0.35.1-kafka-3.4.0` container image.
- For Kafka Exporter:
  1. Container image specified in the `STRIMZI_DEFAULT_KAFKA_EXPORTER_IMAGE` environment variable from the Cluster Operator configuration.

2. `quay.io/strimzi/kafka:0.35.1-kafka-3.4.0` container image.

- For Kafka Bridge:
  1. Container image specified in the `STRIMZI_DEFAULT_KAFKA_BRIDGE_IMAGE` environment variable from the Cluster Operator configuration.
  2. `quay.io/strimzi/kafka-bridge:0.25.0` container image.
- For Kafka broker initializer:
  1. Container image specified in the `STRIMZI_DEFAULT_KAFKA_INIT_IMAGE` environment variable from the Cluster Operator configuration.
  2. `quay.io/strimzi/operator:0.35.1` container image.

Example container image configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    image: my-org/my-image:latest
    # ...
  zookeeper:
    # ...
```

4.1.7. `livenessProbe` and `readinessProbe` healthchecks

Use the `livenessProbe` and `readinessProbe` properties to configure healthcheck probes supported in Strimzi.

Healthchecks are periodical tests which verify the health of an application. When a Healthcheck probe fails, Kubernetes assumes that the application is not healthy and attempts to fix it.

For more details about the probes, see Configure Liveness and Readiness Probes.

Both `livenessProbe` and `readinessProbe` support the following options:

- `initialDelaySeconds`
- `timeoutSeconds`
- `periodSeconds`
- `successThreshold`
- `failureThreshold`

Example of liveness and readiness probe configuration

```
# ...
readinessProbe:
```

```
  initialDelaySeconds: 15
  timeoutSeconds: 5
livenessProbe:
  initialDelaySeconds: 15
  timeoutSeconds: 5
# ...
```

For more information about the `livenessProbe` and `readinessProbe` options, see the Probe schema reference.

4.1.8. `metricsConfig`

Use the `metricsConfig` property to enable and configure Prometheus metrics.

The `metricsConfig` property contains a reference to a ConfigMap that has additional configurations for the Prometheus JMX Exporter. Strimzi supports Prometheus metrics using Prometheus JMX exporter to convert the JMX metrics supported by Apache Kafka and ZooKeeper to Prometheus metrics.

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

Example ConfigMap with metrics configuration for Kafka

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

Example metrics configuration for Kafka

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

```
spec:
  kafka:
    # ...
    metricsConfig:
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: my-config-map
          key: my-key
    # ...
  zookeeper:
    # ...
```

When metrics are enabled, they are exposed on port 9404.

When the `metricsConfig` (or deprecated `metrics`) property is not defined in the resource, the Prometheus metrics are disabled.

For more information about setting up and deploying Prometheus and Grafana, see Introducing Metrics to Kafka in the *Deploying and Upgrading Strimzi* guide.

4.1.9. `jvmOptions`

The following Strimzi components run inside a Java Virtual Machine (JVM):

- Apache Kafka
- Apache ZooKeeper
- Apache Kafka Connect
- Apache Kafka MirrorMaker
- Strimzi Kafka Bridge

To optimize their performance on different platforms and architectures, you configure the `jvmOptions` property in the following resources:

- `Kafka.spec.kafka`
- `Kafka.spec.zookeeper`
- `Kafka.spec.entityOperator.userOperator`
- `Kafka.spec.entityOperator.topicOperator`
- `Kafka.spec.cruiseControl`
- `KafkaConnect.spec`
- `KafkaMirrorMaker.spec`
- `KafkaMirrorMaker2.spec`
- `KafkaBridge.spec`

You can specify the following options in your configuration:

`-Xms`
: Minimum initial allocation heap size when the JVM starts

`-Xmx`
: Maximum heap size

`-XX`
: Advanced runtime options for the JVM

`javaSystemProperties`
: Additional system properties

`gcLoggingEnabled`
: Enables garbage collector logging

NOTE The units accepted by JVM settings, such as `-Xmx` and `-Xms`, are the same units accepted by the JDK `java` binary in the corresponding image. Therefore, `1g` or `1G` means 1,073,741,824 bytes, and `Gi` is not a valid unit suffix. This is different from the units used for memory requests and limits, which follow the Kubernetes convention where `1G` means 1,000,000,000 bytes, and `1Gi` means 1,073,741,824 bytes.

`-Xms` and `-Xmx` options

In addition to setting memory request and limit values for your containers, you can use the `-Xms` and `-Xmx` JVM options to set specific heap sizes for your JVM. Use the `-Xms` option to set an initial heap size and the `-Xmx` option to set a maximum heap size.

Specify heap size to have more control over the memory allocated to your JVM. Heap sizes should make the best use of a container's memory limit (and request) without exceeding it. Heap size and any other memory requirements need to fit within a specified memory limit. If you don't specify heap size in your configuration, but you configure a memory resource limit (and request), the Cluster Operator imposes default heap sizes automatically. The Cluster Operator sets default maximum and minimum heap values based on a percentage of the memory resource configuration.

The following table shows the default heap values.

Table 6. Default heap settings for components

Component	Percent of available memory allocated to the heap	Maximum limit
Kafka	50%	5 GB
ZooKeeper	75%	2 GB
Kafka Connect	75%	None
MirrorMaker 2	75%	None
MirrorMaker	75%	None

Component	Percent of available memory allocated to the heap	Maximum limit
Cruise Control	75%	None
Kafka Bridge	50%	31 Gi

If a memory limit (and request) is not specified, a JVM's minimum heap size is set to `128M`. The JVM's maximum heap size is not defined to allow the memory to increase as needed. This is ideal for single node environments in test and development.

Setting an appropriate memory request can prevent the following:

- Kubernetes killing a container if there is pressure on memory from other pods running on the node.
- Kubernetes scheduling a container to a node with insufficient memory. If `-Xms` is set to `-Xmx`, the container will crash immediately; if not, the container will crash at a later time.

In this example, the JVM uses 2 GiB (=2,147,483,648 bytes) for its heap. Total JVM memory usage can be a lot more than the maximum heap size.

Example `-Xmx` and `-Xms` configuration

```
# ...
jvmOptions:
  "-Xmx": "2g"
  "-Xms": "2g"
# ...
```

Setting the same value for initial (`-Xms`) and maximum (`-Xmx`) heap sizes avoids the JVM having to allocate memory after startup, at the cost of possibly allocating more heap than is really needed.

IMPORTANT

Containers performing lots of disk I/O, such as Kafka broker containers, require available memory for use as an operating system page cache. For such containers, the requested memory should be significantly higher than the memory used by the JVM.

-XX option

`-XX` options are used to configure the `KAFKA_JVM_PERFORMANCE_OPTS` option of Apache Kafka.

Example `-XX` configuration

```
jvmOptions:
  "-XX":
    "UseG1GC": true
    "MaxGCPauseMillis": 20
    "InitiatingHeapOccupancyPercent": 35
    "ExplicitGCInvokesConcurrent": true
```

JVM options resulting from the `-XX` configuration

```
-XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35
-XX:+ExplicitGCInvokesConcurrent -XX:-UseParNewGC
```

NOTE	When no `-XX` options are specified, the default Apache Kafka configuration of `KAFKA_JVM_PERFORMANCE_OPTS` is used.

`javaSystemProperties`

`javaSystemProperties` are used to configure additional Java system properties, such as debugging utilities.

Example `javaSystemProperties` configuration

```
jvmOptions:
  javaSystemProperties:
    - name: javax.net.debug
      value: ssl
```

For more information about the `jvmOptions`, see the `JvmOptions` schema reference.

4.1.10. Garbage collector logging

The `jvmOptions` property also allows you to enable and disable garbage collector (GC) logging. GC logging is disabled by default. To enable it, set the `gcLoggingEnabled` property as follows:

Example GC logging configuration

```
# ...
jvmOptions:
  gcLoggingEnabled: true
# ...
```

4.2. Schema properties

4.2.1. `Kafka` schema reference

Property	Description
spec `KafkaSpec`	The specification of the Kafka and ZooKeeper clusters, and Topic Operator.
status `KafkaStatus`	The status of the Kafka and ZooKeeper clusters, and Topic Operator.

4.2.2. `KafkaSpec` schema reference

Used in: `Kafka`

Property	Description
kafka `KafkaClusterSpec`	Configuration of the Kafka cluster.
zookeeper `ZookeeperClusterSpec`	Configuration of the ZooKeeper cluster.
entityOperator `EntityOperatorSpec`	Configuration of the Entity Operator.
clusterCa `CertificateAuthority`	Configuration of the cluster certificate authority.
clientsCa `CertificateAuthority`	Configuration of the clients certificate authority.
cruiseControl `CruiseControlSpec`	Configuration for Cruise Control deployment. Deploys a Cruise Control instance when specified.
jmxTrans `JmxTransSpec`	**The `jmxTrans` property has been deprecated.** JMXTrans is deprecated and related resources removed in Strimzi 0.35.0. As of Strimzi 0.35.0, JMXTrans is not supported anymore and this option is ignored.
kafkaExporter `KafkaExporterSpec`	Configuration of the Kafka Exporter. Kafka Exporter can provide additional metrics, for example lag of consumer group at topic/partition.
maintenanceTimeWindows string array	A list of time windows for maintenance tasks (that is, certificates renewal). Each time window is defined by a cron expression.

4.2.3. `KafkaClusterSpec` schema reference

Used in: `KafkaSpec`

Full list of `KafkaClusterSpec` schema properties

Configures a Kafka cluster.

`listeners`

Use the `listeners` property to configure listeners to provide access to Kafka brokers.

Example configuration of a plain (unencrypted) listener without authentication

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
    # ...
  zookeeper:
    # ...
```

`config`

Use the `config` properties to configure Kafka broker options as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the Apache Kafka documentation.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Security (encryption, authentication, and authorization)
- Listener configuration
- Broker ID configuration
- Configuration of log data directories
- Inter-broker communication
- ZooKeeper connectivity

Properties with the following prefixes cannot be set:

- `advertised.`
- `authorizer.`
- `broker.`
- `controller`

- `cruise.control.metrics.reporter.bootstrap.`
- `cruise.control.metrics.topic`
- `host.name`
- `inter.broker.listener.name`
- `listener.`
- `listeners.`
- `log.dir`
- `password.`
- `port`
- `process.roles`
- `sasl.`
- `security.`
- `servers,node.id`
- `ssl.`
- `super.user`
- `zookeeper.clientCnxnSocket`
- `zookeeper.connect`
- `zookeeper.set.acl`
- `zookeeper.ssl`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to Kafka, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites
- Configuration for the `zookeeper.connection.timeout.ms` property to set the maximum time allowed for establishing a ZooKeeper connection
- Cruise Control metrics properties:
  - `cruise.control.metrics.topic.num.partitions`
  - `cruise.control.metrics.topic.replication.factor`
  - `cruise.control.metrics.topic.retention.ms`
  - `cruise.control.metrics.topic.auto.create.retries`
  - `cruise.control.metrics.topic.auto.create.timeout.ms`
  - `cruise.control.metrics.topic.min.insync.replicas`
- Controller properties:
  - `controller.quorum.election.backoff.max.ms`
  - `controller.quorum.election.timeout.ms`

- `controller.quorum.fetch.timeout.ms`

Example Kafka broker configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      num.partitions: 1
      num.recovery.threads.per.data.dir: 1
      default.replication.factor: 3
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 1
      log.retention.hours: 168
      log.segment.bytes: 1073741824
      log.retention.check.interval.ms: 300000
      num.network.threads: 3
      num.io.threads: 8
      socket.send.buffer.bytes: 102400
      socket.receive.buffer.bytes: 102400
      socket.request.max.bytes: 104857600
      group.initial.rebalance.delay.ms: 0
      zookeeper.connection.timeout.ms: 6000
    # ...
```

`brokerRackInitImage`

When rack awareness is enabled, Kafka broker pods use init container to collect the labels from the Kubernetes cluster nodes. The container image used for this container can be configured using the `brokerRackInitImage` property. When the `brokerRackInitImage` field is missing, the following images are used in order of priority:

1. Container image specified in `STRIMZI_DEFAULT_KAFKA_INIT_IMAGE` environment variable in the Cluster Operator configuration.
2. `quay.io/strimzi/operator:0.35.1` container image.

Example `brokerRackInitImage` configuration

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

```
    rack:
      topologyKey: topology.kubernetes.io/zone
    brokerRackInitImage: my-org/my-image:latest
    # ...
```

NOTE	Overriding container images is recommended only in special situations, where you need to use a different container registry. For example, because your network does not allow access to the container registry used by Strimzi. In this case, you should either copy the Strimzi images or build them from the source. If the configured image is not compatible with Strimzi images, it might not work properly.

`logging`

Kafka has its own configurable loggers:

- `log4j.logger.org.I0Itec.zkclient.ZkClient`
- `log4j.logger.org.apache.zookeeper`
- `log4j.logger.kafka`
- `log4j.logger.org.apache.kafka`
- `log4j.logger.kafka.request.logger`
- `log4j.logger.kafka.network.Processor`
- `log4j.logger.kafka.server.KafkaApis`
- `log4j.logger.kafka.network.RequestChannel$`
- `log4j.logger.kafka.controller`
- `log4j.logger.kafka.log.LogCleaner`
- `log4j.logger.state.change.logger`
- `log4j.logger.kafka.authorizer.logger`

Kafka uses the Apache `log4j` logger implementation.

Use the `logging` property to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. Inside the ConfigMap, the logging configuration is described using `log4j.properties`. Both `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. For more information about log levels, see Apache logging services.

Here we see examples of `inline` and `external` logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  kafka:
    # ...
    logging:
      type: inline
      loggers:
        kafka.root.logger.level: "INFO"
  # ...
```

External logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: customConfigMap
        key: kafka-log4j.properties
  # ...
```

Any available loggers that are not configured have their level set to `OFF`.

If Kafka was deployed using the Cluster Operator, changes to Kafka logging levels are applied dynamically.

If you use external logging, a rolling update is triggered when logging appenders are changed.

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`KafkaClusterSpec` schema properties

Property	Description
version	The kafka broker version. Defaults to 3.4.0. Consult the user documentation to understand the process required to upgrade or downgrade the version.
string	
replicas	The number of pods in the cluster.
integer	

Property	Description
image	The docker image for the pods. The default value depends on the configured `Kafka.spec.kafka.version`.
string	
listeners	Configures listeners of Kafka brokers.
`GenericKafkaListener` array	
config	Kafka broker config properties with the following prefixes cannot be set: listeners, advertised., broker., listener., host.name, port, inter.broker.listener.name, sasl., ssl., security., password., log.dir, zookeeper.connect, zookeeper.set.acl, zookeeper.ssl, zookeeper.clientCnxnSocket, authorizer., super.user, cruise.control.metrics.topic, cruise.control.metrics.reporter.bootstrap.servers,node.id, process.roles, controller. (with the exception of: zookeeper.connection.timeout.ms, sasl.server.max.receive.size,ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols, ssl.secure.random.implementation,cruise.control.metrics.topic.num.partitions, cruise.control.metrics.topic.replication.factor, cruise.control.metrics.topic.retention.ms,cruise.control.metrics.topic.auto.create.retries, cruise.control.metrics.topic.auto.create.timeout.ms,cruise.control.metrics.topic.min.insync.replicas,controller.quorum.election.backoff.max.ms, controller.quorum.election.timeout.ms, controller.quorum.fetch.timeout.ms).
map	
storage	Storage configuration (disk). Cannot be updated. The type depends on the value of the `storage.type` property within the given object, which must be one of [ephemeral, persistent-claim, jbod].
`EphemeralStorage`, `PersistentClaimStorage`, `JbodStorage`	
authorization	Authorization configuration for Kafka brokers. The type depends on the value of the `authorization.type` property within the given object, which must be one of [simple, opa, keycloak, custom].
`KafkaAuthorizationSimple`, `KafkaAuthorizationOpa`, `KafkaAuthorizationKeycloak`, `KafkaAuthorizationCustom`	
rack	Configuration of the `broker.rack` broker config.
`Rack`	
brokerRackInitImage	The image of the init container used for initializing the `broker.rack`.
string	

Property	Description
livenessProbe Probe	Pod liveness checking.
readinessProbe Probe	Pod readiness checking.
jvmOptions JvmOptions	JVM Options for pods.
jmxOptions KafkaJmxOptions	JMX Options for Kafka brokers.
resources ResourceRequirements	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
metricsConfig JmxPrometheusExporterMetrics	Metrics configuration. The type depends on the value of the metricsConfig.type property within the given object, which must be one of [jmxPrometheusExporter].
logging InlineLogging, ExternalLogging	Logging configuration for Kafka. The type depends on the value of the logging.type property within the given object, which must be one of [inline, external].
template KafkaClusterTemplate	Template for Kafka cluster resources. The template allows users to specify how the Kubernetes resources are generated.

4.2.4. GenericKafkaListener schema reference

Used in: KafkaClusterSpec

Full list of GenericKafkaListener schema properties

Configures listeners to connect to Kafka brokers within and outside Kubernetes.

You configure the listeners in the Kafka resource.

Example Kafka resource showing listener configuration

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

```
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
      - name: external1
        port: 9094
        type: route
        tls: true
      - name: external2
        port: 9095
        type: ingress
        tls: true
        authentication:
          type: tls
        configuration:
          bootstrap:
            host: bootstrap.myingress.com
          brokers:
          - broker: 0
            host: broker-0.myingress.com
          - broker: 1
            host: broker-1.myingress.com
          - broker: 2
            host: broker-2.myingress.com
    #...
```

`listeners`

You configure Kafka broker listeners using the `listeners` property in the `Kafka` resource. Listeners are defined as an array.

Example listener configuration

```
listeners:
  - name: plain
    port: 9092
    type: internal
    tls: false
```

The name and port must be unique within the Kafka cluster. The name can be up to 25 characters long, comprising lower-case letters and numbers. Allowed port numbers are 9092 and higher with the exception of ports 9404 and 9999, which are already used for Prometheus and JMX.

By specifying a unique name and port for each listener, you can configure multiple listeners.

`type`

The type is set as `internal`, or for external listeners, as `route`, `loadbalancer`, `nodeport`, `ingress` or `cluster-ip`. You can also configure a `cluster-ip` listener, a type of internal listener you can use to build custom access mechanisms.

internal

You can configure internal listeners with or without encryption using the `tls` property.

Example `internal` listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    #...
```

route

Configures an external listener to expose Kafka using OpenShift `Routes` and the HAProxy router.

A dedicated `Route` is created for every Kafka broker pod. An additional `Route` is created to serve as a Kafka bootstrap address. Kafka clients can use these `Routes` to connect to Kafka on port 443. The client connects on port 443, the default router port, but traffic is then routed to the port you configure, which is `9094` in this example.

Example `route` listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: external1
        port: 9094
        type: route
        tls: true
    #...
```

ingress

Configures an external listener to expose Kafka using Kubernetes `Ingress` and the Ingress NGINX Controller for Kubernetes.

A dedicated `Ingress` resource is created for every Kafka broker pod. An additional `Ingress` resource is created to serve as a Kafka bootstrap address. Kafka clients can use these `Ingress` resources to connect to Kafka on port 443. The client connects on port 443, the default controller port, but traffic is then routed to the port you configure, which is `9095` in the following example.

You must specify the hostnames used by the bootstrap and per-broker services using `GenericKafkaListenerConfigurationBootstrap` and `GenericKafkaListenerConfigurationBroker` properties.

Example `ingress` listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: external2
        port: 9095
        type: ingress
        tls: true
        authentication:
          type: tls
        configuration:
          bootstrap:
            host: bootstrap.myingress.com
          brokers:
          - broker: 0
            host: broker-0.myingress.com
          - broker: 1
            host: broker-1.myingress.com
          - broker: 2
            host: broker-2.myingress.com
  #...
```

NOTE External listeners using `Ingress` are currently only tested with the Ingress NGINX Controller for Kubernetes.

loadbalancer

Configures an external listener to expose Kafka using a `Loadbalancer` type `Service`.

A new loadbalancer service is created for every Kafka broker pod. An additional loadbalancer is created to serve as a Kafka *bootstrap* address. Loadbalancers listen to the specified port number, which is port `9094` in the following example.

You can use the `loadBalancerSourceRanges` property to configure source ranges to restrict access to the specified IP addresses.

Example `loadbalancer` listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      - name: external3
        port: 9094
        type: loadbalancer
        tls: true
        configuration:
          loadBalancerSourceRanges:
            - 10.0.0.0/8
            - 88.208.76.87/32
    #...
```

nodeport

Configures an external listener to expose Kafka using a `NodePort` type `Service`.

Kafka clients connect directly to the nodes of Kubernetes. An additional `NodePort` type of service is created to serve as a Kafka bootstrap address.

When configuring the advertised addresses for the Kafka broker pods, Strimzi uses the address of the node on which the given pod is running. You can use `preferredNodePortAddressType` property to configure the first address type checked as the node address.

Example `nodeport` listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      #...
      - name: external4
        port: 9095
        type: nodeport
        tls: false
        configuration:
          preferredNodePortAddressType: InternalDNS
    #...
```

NOTE TLS hostname verification is not currently supported when exposing Kafka clusters using node ports.

cluster-ip

Configures an internal listener to expose Kafka using a per-broker `ClusterIP` type `Service`.

The listener does not use a headless service and its DNS names to route traffic to Kafka brokers. You can use this type of listener to expose a Kafka cluster when using the headless service is unsuitable. You might use it with a custom access mechanism, such as one that uses a specific Ingress controller or the Kubernetes Gateway API.

A new `ClusterIP` service is created for each Kafka broker pod. The service is assigned a `ClusterIP` address to serve as a Kafka *bootstrap* address with a per-broker port number. For example, you can configure the listener to expose a Kafka cluster over an Nginx Ingress Controller with TCP port configuration.

Example `cluster-ip` listener configuration

```
#...
spec:
  kafka:
    #...
    listeners:
      - name: external-cluster-ip
        type: cluster-ip
        tls: false
        port: 9096
    #...
```

`port`

The port number is the port used in the Kafka cluster, which might not be the same port used for access by a client.

- `loadbalancer` listeners use the specified port number, as do `internal` and `cluster-ip` listeners
- `ingress` and `route` listeners use port 443 for access
- `nodeport` listeners use the port number assigned by Kubernetes

For client connection, use the address and port for the bootstrap service of the listener. You can retrieve this from the status of the `Kafka` resource.

Example command to retrieve the address and port for client connection

```
kubectl get kafka <kafka_cluster_name> -
o=jsonpath='{.status.listeners[?(@.name=="<listener_name>")].bootstrapServers}{"\n"}'
```

NOTE Listeners cannot be configured to use the ports set aside for interbroker communication (9090 and 9091) and metrics (9404).

`tls`

The TLS property is required.

By default, TLS encryption is not enabled. To enable it, set the `tls` property to `true`.

For `route` and `ingress` type listeners, TLS encryption must be enabled.

`authentication`

Authentication for the listener can be specified as:

- mTLS (`tls`)
- SCRAM-SHA-512 (`scram-sha-512`)
- Token-based OAuth 2.0 (`oauth`)
- Custom (`custom`)

`networkPolicyPeers`

Use `networkPolicyPeers` to configure network policies that restrict access to a listener at the network level. The following example shows a `networkPolicyPeers` configuration for a `plain` and a `tls` listener.

In the following example:

- Only application pods matching the labels `app: kafka-sasl-consumer` and `app: kafka-sasl-producer` can connect to the `plain` listener. The application pods must be running in the same namespace as the Kafka broker.
- Only application pods running in namespaces matching the labels `project: myproject` and `project: myproject2` can connect to the `tls` listener.

The syntax of the `networkPolicyPeers` property is the same as the `from` property in `NetworkPolicy` resources.

Exanmple network policy configuration

```
listeners:
  #...
  - name: plain
    port: 9092
    type: internal
    tls: true
    authentication:
      type: scram-sha-512
    networkPolicyPeers:
      - podSelector:
          matchLabels:
            app: kafka-sasl-consumer
      - podSelector:
          matchLabels:
            app: kafka-sasl-producer
```

```
  - name: tls
    port: 9093
    type: internal
    tls: true
    authentication:
      type: tls
    networkPolicyPeers:
      - namespaceSelector:
          matchLabels:
            project: myproject
      - namespaceSelector:
          matchLabels:
            project: myproject2
# ...
```

`GenericKafkaListener` **schema properties**

Property	Description
name	Name of the listener. The name will be used to identify the listener and the related Kubernetes objects. The name has to be unique within given a Kafka cluster. The name can consist of lowercase characters and numbers and be up to 11 characters long.
string	
port	Port number used by the listener inside Kafka. The port number has to be unique within a given Kafka cluster. Allowed port numbers are 9092 and higher with the exception of ports 9404 and 9999, which are already used for Prometheus and JMX. Depending on the listener type, the port number might not be the same as the port number that connects Kafka clients.
integer	

Property	Description
type	Type of the listener. Currently the supported types are `internal`, `route`, `loadbalancer`, `nodeport` and `ingress`. • `internal` type exposes Kafka internally only within the Kubernetes cluster. • `route` type uses OpenShift Routes to expose Kafka. • `loadbalancer` type uses LoadBalancer type services to expose Kafka. • `nodeport` type uses NodePort type services to expose Kafka. • `ingress` type uses Kubernetes Nginx Ingress to expose Kafka with TLS passthrough. • `cluster-ip` type uses a per-broker `ClusterIP` service.
string (one of [ingress, internal, route, loadbalancer, cluster-ip, nodeport])	
tls	Enables TLS encryption on the listener. This is a required property.
boolean	
authentication	Authentication configuration for this listener. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, scram-sha-512, oauth, custom].
`KafkaListenerAuthenticationTls`, `KafkaListenerAuthenticationScramSha512`, `KafkaListenerAuthenticationOAuth`, `KafkaListenerAuthenticationCustom`	
configuration	Additional listener configuration.
`GenericKafkaListenerConfiguration`	
networkPolicyPeers	List of peers which should be able to connect to this listener. Peers in this list are combined using a logical OR operation. If this field is empty or missing, all connections will be allowed for this listener. If this field is present and contains at least one item, the listener only allows the traffic which matches at least one item in this list. For more information, see the external documentation for networking.k8s.io/v1 networkpolicypeer.
NetworkPolicyPeer array	

4.2.5. `KafkaListenerAuthenticationTls` schema reference

Used in: `GenericKafkaListener`

The `type` property is a discriminator that distinguishes use of the `KafkaListenerAuthenticationTls`

type from KafkaListenerAuthenticationScramSha512, KafkaListenerAuthenticationOAuth, KafkaListenerAuthenticationCustom. It must have the value tls for the type KafkaListenerAuthenticationTls.

Property	Description
type	Must be tls.
string	

4.2.6. KafkaListenerAuthenticationScramSha512 schema reference

Used in: GenericKafkaListener

The type property is a discriminator that distinguishes use of the KafkaListenerAuthenticationScramSha512 type from KafkaListenerAuthenticationTls, KafkaListenerAuthenticationOAuth, KafkaListenerAuthenticationCustom. It must have the value scram-sha-512 for the type KafkaListenerAuthenticationScramSha512.

Property	Description
type	Must be scram-sha-512.
string	

4.2.7. KafkaListenerAuthenticationOAuth schema reference

Used in: GenericKafkaListener

The type property is a discriminator that distinguishes use of the KafkaListenerAuthenticationOAuth type from KafkaListenerAuthenticationTls, KafkaListenerAuthenticationScramSha512, KafkaListenerAuthenticationCustom. It must have the value oauth for the type KafkaListenerAuthenticationOAuth.

Property	Description
accessTokenIsJwt	Configure whether the access token is treated as JWT. This must be set to false if the authorization server returns opaque tokens. Defaults to true.
boolean	
checkAccessTokenType	Configure whether the access token type check is performed or not. This should be set to false if the authorization server does not include 'typ' claim in JWT token. Defaults to true.
boolean	

Property	Description
checkAudience	Enable or disable audience checking. Audience checks identify the recipients of tokens. If audience checking is enabled, the OAuth Client ID also has to be configured using the `clientId` property. The Kafka broker will reject tokens that do not have its `clientId` in their `aud` (audience) claim.Default value is `false`.
boolean	
checkIssuer	Enable or disable issuer checking. By default issuer is checked using the value configured by `validIssuerUri`. Default value is `true`.
boolean	
clientAudience	The audience to use when making requests to the authorization server's token endpoint. Used for inter-broker authentication and for configuring OAuth 2.0 over PLAIN using the `clientId` and `secret` method.
string	
clientId	OAuth Client ID which the Kafka broker can use to authenticate against the authorization server and use the introspect endpoint URI.
string	
clientScope	The scope to use when making requests to the authorization server's token endpoint. Used for inter-broker authentication and for configuring OAuth 2.0 over PLAIN using the `clientId` and `secret` method.
string	
clientSecret	Link to Kubernetes Secret containing the OAuth client secret which the Kafka broker can use to authenticate against the authorization server and use the introspect endpoint URI.
`GenericSecretSource`	
connectTimeoutSeconds	The connect timeout in seconds when connecting to authorization server. If not set, the effective connect timeout is 60 seconds.
integer	
customClaimCheck	JsonPath filter query to be applied to the JWT token or to the response of the introspection endpoint for additional token validation. Not set by default.
string	
disableTlsHostnameVerification	Enable or disable TLS hostname verification. Default value is `false`.
boolean	
enableECDSA	**The `enableECDSA` property has been deprecated.** Enable or disable ECDSA support by installing BouncyCastle crypto provider. ECDSA support is always enabled. The BouncyCastle libraries are no longer packaged with Strimzi. Value is ignored.
boolean	

Property	Description
enableMetrics	Enable or disable OAuth metrics. Default value is `false`.
boolean	
enableOauthBearer	Enable or disable OAuth authentication over SASL_OAUTHBEARER. Default value is `true`.
boolean	
enablePlain	Enable or disable OAuth authentication over SASL_PLAIN. There is no re-authentication support when this mechanism is used. Default value is `false`.
boolean	
failFast	Enable or disable termination of Kafka broker processes due to potentially recoverable runtime errors during startup. Default value is `true`.
boolean	
fallbackUserNameClaim	The fallback username claim to be used for the user id if the claim specified by `userNameClaim` is not present. This is useful when `client_credentials` authentication only results in the client id being provided in another claim. It only takes effect if `userNameClaim` is set.
string	
fallbackUserNamePrefix	The prefix to use with the value of `fallbackUserNameClaim` to construct the user id. This only takes effect if `fallbackUserNameClaim` is true, and the value is present for the claim. Mapping usernames and client ids into the same user id space is useful in preventing name collisions.
string	
groupsClaim	JsonPath query used to extract groups for the user during authentication. Extracted groups can be used by a custom authorizer. By default no groups are extracted.
string	
groupsClaimDelimiter	A delimiter used to parse groups when they are extracted as a single String value rather than a JSON array. Default value is ',' (comma).
string	
httpRetries	The maximum number of retries to attempt if an initial HTTP request fails. If not set, the default is to not attempt any retries.
integer	
httpRetryPauseMs	The pause to take before retrying a failed HTTP request. If not set, the default is to not pause at all but to immediately repeat a request.
integer	
introspectionEndpointUri	URI of the token introspection endpoint which can be used to validate opaque non-JWT tokens.
string	

Property	Description
jwksEndpointUri string	URI of the JWKS certificate endpoint, which can be used for local JWT validation.
jwksExpirySeconds integer	Configures how often are the JWKS certificates considered valid. The expiry interval has to be at least 60 seconds longer then the refresh interval specified in `jwksRefreshSeconds`. Defaults to 360 seconds.
jwksIgnoreKeyUse boolean	Flag to ignore the 'use' attribute of `key` declarations in a JWKS endpoint response. Default value is `false`.
jwksMinRefreshPauseSeconds integer	The minimum pause between two consecutive refreshes. When an unknown signing key is encountered the refresh is scheduled immediately, but will always wait for this minimum pause. Defaults to 1 second.
jwksRefreshSeconds integer	Configures how often are the JWKS certificates refreshed. The refresh interval has to be at least 60 seconds shorter then the expiry interval specified in `jwksExpirySeconds`. Defaults to 300 seconds.
maxSecondsWithoutReauthentication integer	Maximum number of seconds the authenticated session remains valid without re-authentication. This enables Apache Kafka re-authentication feature, and causes sessions to expire when the access token expires. If the access token expires before max time or if max time is reached, the client has to re-authenticate, otherwise the server will drop the connection. Not set by default - the authenticated session does not expire when the access token expires. This option only applies to SASL_OAUTHBEARER authentication mechanism (when `enableOauthBearer` is `true`).
readTimeoutSeconds integer	The read timeout in seconds when connecting to authorization server. If not set, the effective read timeout is 60 seconds.
tlsTrustedCertificates `CertSecretSource` array	Trusted certificates for TLS connection to the OAuth server.

Property	Description
tokenEndpointUri	URI of the Token Endpoint to use with SASL_PLAIN mechanism when the client authenticates with `clientId` and a `secret`. If set, the client can authenticate over SASL_PLAIN by either setting `username` to `clientId`, and setting `password` to client `secret`, or by setting `username` to account username, and `password` to access token prefixed with `$accessToken:`. If this option is not set, the `password` is always interpreted as an access token (without a prefix), and `username` as the account username (a so called 'no-client-credentials' mode).
string	
type	Must be `oauth`.
string	
userInfoEndpointUri	URI of the User Info Endpoint to use as a fallback to obtaining the user id when the Introspection Endpoint does not return information that can be used for the user id.
string	
userNameClaim	Name of the claim from the JWT authentication token, Introspection Endpoint response or User Info Endpoint response which will be used to extract the user id. Defaults to `sub`.
string	
validIssuerUri	URI of the token issuer used for authentication.
string	
validTokenType	Valid value for the `token_type` attribute returned by the Introspection Endpoint. No default value, and not checked by default.
string	

4.2.8. `GenericSecretSource` schema reference

Used in: `KafkaClientAuthenticationOAuth`, `KafkaListenerAuthenticationCustom`, `KafkaListenerAuthenticationOAuth`

Property	Description
key	The key under which the secret value is stored in the Kubernetes Secret.
string	
secretName	The name of the Kubernetes Secret containing the secret value.
string	

4.2.9. `CertSecretSource` schema reference

Used in: `ClientTls`, `KafkaAuthorizationKeycloak`, `KafkaAuthorizationOpa`,

KafkaClientAuthenticationOAuth, KafkaListenerAuthenticationOAuth

Property	Description
certificate	The name of the file certificate in the Secret.
string	
secretName	The name of the Secret containing the certificate.
string	

4.2.10. KafkaListenerAuthenticationCustom schema reference

Used in: GenericKafkaListener

Full list of KafkaListenerAuthenticationCustom schema properties

To configure custom authentication, set the type property to custom.

Custom authentication allows for any type of kafka-supported authentication to be used.

Example custom OAuth authentication configuration

```
spec:
  kafka:
    config:
      principal.builder.class: SimplePrincipal.class
    listeners:
      - name: oauth-bespoke
        port: 9093
        type: internal
        tls: true
        authentication:
          type: custom
          sasl: true
          listenerConfig:
            oauthbearer.sasl.client.callback.handler.class: client.class
            oauthbearer.sasl.server.callback.handler.class: server.class
            oauthbearer.sasl.login.callback.handler.class: login.class
            oauthbearer.connections.max.reauth.ms: 999999999
            sasl.enabled.mechanisms: oauthbearer
            oauthbearer.sasl.jaas.config: |
              org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule
required ;
          secrets:
            - name: example
```

A protocol map is generated that uses the sasl and tls values to determine which protocol to map to the listener.

- SASL = True, TLS = True → SASL_SSL

- SASL = False, TLS = True → SSL
- SASL = True, TLS = False → SASL_PLAINTEXT
- SASL = False, TLS = False → PLAINTEXT

`listenerConfig`

Listener configuration specified using `listenerConfig` is prefixed with `listener.name.<listener_name>-<port>`. For example, `sasl.enabled.mechanisms` becomes `listener.name.<listener_name>-<port>.sasl.enabled.mechanisms`.

`secrets`

Secrets are mounted to `/opt/kafka/custom-authn-secrets/custom-listener-<listener_name>-<port>/<secret_name>` in the Kafka broker nodes' containers.

For example, the mounted secret (`example`) in the example configuration would be located at `/opt/kafka/custom-authn-secrets/custom-listener-oauth-bespoke-9093/example`.

Principal builder

You can set a custom principal builder in the Kafka cluster configuration. However, the principal builder is subject to the following requirements:

- The specified principal builder class must exist on the image. *Before* building your own, check if one already exists. You'll need to rebuild the Strimzi images with the required classes.
- No other listener is using `oauth` type authentication. This is because an OAuth listener appends its own principle builder to the Kafka configuration.
- The specified principal builder is compatible with Strimzi.

Custom principal builders must support peer certificates for authentication, as Strimzi uses these to manage the Kafka cluster.

A custom OAuth principal builder might be identical or very similar to the Strimzi OAuth principal builder.

NOTE Kafka's default principal builder class supports the building of principals based on the names of peer certificates. The custom principal builder should provide a principal of type `user` using the name of the SSL peer certificate.

The following example shows a custom principal builder that satisfies the OAuth requirements of Strimzi.

Example principal builder for custom OAuth configuration

```
public final class CustomKafkaPrincipalBuilder implements KafkaPrincipalBuilder {

    public KafkaPrincipalBuilder() {}

    @Override
```

```
    public KafkaPrincipal build(AuthenticationContext context) {
        if (context instanceof SslAuthenticationContext) {
            SSLSession sslSession = ((SslAuthenticationContext) context).session();
            try {
                return new KafkaPrincipal(
                    KafkaPrincipal.USER_TYPE,
sslSession.getPeerPrincipal().getName());
            } catch (SSLPeerUnverifiedException e) {
                throw new IllegalArgumentException("Cannot use an unverified peer for
authentication", e);
            }
        }

        // Create your own KafkaPrincipal here
        ...
    }
}
```

`KafkaListenerAuthenticationCustom` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaListenerAuthenticationCustom` type from `KafkaListenerAuthenticationTls`, `KafkaListenerAuthenticationScramSha512`, `KafkaListenerAuthenticationOAuth`. It must have the value `custom` for the type `KafkaListenerAuthenticationCustom`.

Property	Description
listenerConfig	Configuration to be used for a specific listener. All values are prefixed with listener.name.<*listener_name*>.
map	
sasl	Enable or disable SASL on this listener.
boolean	
secrets	Secrets to be mounted to /opt/kafka/custom-authn-secrets/custom-listener-<*listener_name*>-<*port*>/<*secret_name*>.
`GenericSecretSource` array	
type	Must be `custom`.
string	

4.2.11. `GenericKafkaListenerConfiguration` schema reference

Used in: `GenericKafkaListener`

Full list of `GenericKafkaListenerConfiguration` schema properties

Configuration for Kafka listeners.

brokerCertChainAndKey

The `brokerCertChainAndKey` property is only used with listeners that have TLS encryption enabled. You can use the property to provide your own Kafka listener certificates.

Example configuration for a `loadbalancer` external listener with TLS encryption enabled

```
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      brokerCertChainAndKey:
        secretName: my-secret
        certificate: my-listener-certificate.crt
        key: my-listener-key.key
# ...
```

externalTrafficPolicy

The `externalTrafficPolicy` property is used with `loadbalancer` and `nodeport` listeners. When exposing Kafka outside of Kubernetes you can choose `Local` or `Cluster`. `Local` avoids hops to other nodes and preserves the client IP, whereas `Cluster` does neither. The default is `Cluster`.

loadBalancerSourceRanges

The `loadBalancerSourceRanges` property is only used with `loadbalancer` listeners. When exposing Kafka outside of Kubernetes use source ranges, in addition to labels and annotations, to customize how a service is created.

Example source ranges configured for a loadbalancer listener

```
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: false
    configuration:
      externalTrafficPolicy: Local
      loadBalancerSourceRanges:
        - 10.0.0.0/8
        - 88.208.76.87/32
      # ...
# ...
```

`class`

The `class` property is only used with `ingress` listeners. You can configure the `Ingress` class using the `class` property.

Example of an external listener of type `ingress` using `Ingress` class `nginx-internal`

```
listeners:
  #...
  - name: external
    port: 9094
    type: ingress
    tls: true
    configuration:
      class: nginx-internal
    # ...
# ...
```

`preferredNodePortAddressType`

The `preferredNodePortAddressType` property is only used with `nodeport` listeners.

Use the `preferredNodePortAddressType` property in your listener configuration to specify the first address type checked as the node address. This property is useful, for example, if your deployment does not have DNS support, or you only want to expose a broker internally through an internal DNS or IP address. If an address of this type is found, it is used. If the preferred address type is not found, Strimzi proceeds through the types in the standard order of priority:

1. ExternalDNS
2. ExternalIP
3. Hostname
4. InternalDNS
5. InternalIP

Example of an external listener configured with a preferred node port address type

```
listeners:
  #...
  - name: external
    port: 9094
    type: nodeport
    tls: false
    configuration:
      preferredNodePortAddressType: InternalDNS
      # ...
# ...
```

`useServiceDnsDomain`

The `useServiceDnsDomain` property is only used with `internal` and `cluster-ip` listeners. It defines whether the fully-qualified DNS names that include the cluster service suffix (usually `.cluster.local`) are used. With `useServiceDnsDomain` set as `false`, the advertised addresses are generated without the service suffix; for example, `my-cluster-kafka-0.my-cluster-kafka-brokers.myproject.svc`. With `useServiceDnsDomain` set as `true`, the advertised addresses are generated with the service suffix; for example, `my-cluster-kafka-0.my-cluster-kafka-brokers.myproject.svc.cluster.local`. Default is `false`.

Example of an internal listener configured to use the Service DNS domain

```
listeners:
  #...
  - name: plain
    port: 9092
    type: internal
    tls: false
    configuration:
      useServiceDnsDomain: true
      # ...
# ...
```

If your Kubernetes cluster uses a different service suffix than `.cluster.local`, you can configure the suffix using the `KUBERNETES_SERVICE_DNS_DOMAIN` environment variable in the Cluster Operator configuration.

`GenericKafkaListenerConfiguration` schema properties

Property	Description
brokerCertChainAndKey	Reference to the `Secret` which holds the certificate and private key pair which will be used for this listener. The certificate can optionally contain the whole chain. This field can be used only with listeners with enabled TLS encryption.
`CertAndKeySecretSource`	
externalTrafficPolicy	Specifies whether the service routes external traffic to node-local or cluster-wide endpoints. `Cluster` may cause a second hop to another node and obscures the client source IP. `Local` avoids a second hop for LoadBalancer and Nodeport type services and preserves the client source IP (when supported by the infrastructure). If unspecified, Kubernetes will use `Cluster` as the default.This field can be used only with `loadbalancer` or `nodeport` type listener.
string (one of [Local, Cluster])	

Property	Description
loadBalancerSourceRanges	A list of CIDR ranges (for example `10.0.0.0/8` or `130.211.204.1/32`) from which clients can connect to load balancer type listeners. If supported by the platform, traffic through the loadbalancer is restricted to the specified CIDR ranges. This field is applicable only for loadbalancer type services and is ignored if the cloud provider does not support the feature. This field can be used only with `loadbalancer` type listener.
string array	
bootstrap	Bootstrap configuration.
`GenericKafkaListenerConfigurationBootstrap`	
brokers	Per-broker configurations.
`GenericKafkaListenerConfigurationBroker` array	
ipFamilyPolicy	Specifies the IP Family Policy used by the service. Available options are `SingleStack`, `PreferDualStack` and `RequireDualStack`. `SingleStack` is for a single IP family. `PreferDualStack` is for two IP families on dual-stack configured clusters or a single IP family on single-stack clusters. `RequireDualStack` fails unless there are two IP families on dual-stack configured clusters. If unspecified, Kubernetes will choose the default value based on the service type. Available on Kubernetes 1.20 and newer.
string (one of [RequireDualStack, SingleStack, PreferDualStack])	
ipFamilies	Specifies the IP Families used by the service. Available options are `IPv4` and `IPv6`. `If unspecified, Kubernetes will choose the default value based on the `ipFamilyPolicy` setting. Available on Kubernetes 1.20 and newer.
string (one or more of [IPv6, IPv4]) array	
createBootstrapService	Whether to create the bootstrap service or not. The bootstrap service is created by default (if not specified differently). This field can be used with the `loadBalancer` type listener.
boolean	

Property	Description
class	Configures a specific class for `Ingress` and `LoadBalancer` that defines which controller will be used. This field can only be used with `ingress` and `loadbalancer` type listeners. If not specified, the default controller is used. For an `ingress` listener, set the `ingressClassName` property in the `Ingress` resources. For a `loadbalancer` listener, set the `loadBalancerClass` property in the `Service` resources.
string	
finalizers	A list of finalizers which will be configured for the `LoadBalancer` type Services created for this listener. If supported by the platform, the finalizer `service.kubernetes.io/load-balancer-cleanup` to make sure that the external load balancer is deleted together with the service.For more information, see https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/#garbage-collecting-load-balancers. This field can be used only with `loadbalancer` type listeners.
string array	
maxConnectionCreationRate	The maximum connection creation rate we allow in this listener at any time. New connections will be throttled if the limit is reached.
integer	
maxConnections	The maximum number of connections we allow for this listener in the broker at any time. New connections are blocked if the limit is reached.
integer	

Property	Description
preferredNodePortAddressType	Defines which address type should be used as the node address. Available types are: `ExternalDNS`, `ExternalIP`, `InternalDNS`, `InternalIP` and `Hostname`. By default, the addresses will be used in the following order (the first one found will be used): • `ExternalDNS` • `ExternalIP` • `InternalDNS` • `InternalIP` • `Hostname` This field is used to select the preferred address type, which is checked first. If no address is found for this address type, the other types are checked in the default order. This field can only be used with `nodeport` type listener.
string (one of [ExternalDNS, ExternalIP, Hostname, InternalIP, InternalDNS])	
useServiceDnsDomain	Configures whether the Kubernetes service DNS domain should be used or not. If set to `true`, the generated addresses will contain the service DNS domain suffix (by default `.cluster.local`, can be configured using environment variable `KUBERNETES_SERVICE_DNS_DOMAIN`). Defaults to `false`.This field can be used only with `internal` and `cluster-ip` type listeners.
boolean	

4.2.12. `CertAndKeySecretSource` schema reference

Used in: `GenericKafkaListenerConfiguration`, `KafkaClientAuthenticationTls`

Property	Description
certificate	The name of the file certificate in the Secret.
string	
key	The name of the private key in the Secret.
string	
secretName	The name of the Secret containing the certificate.
string	

4.2.13. `GenericKafkaListenerConfigurationBootstrap` schema reference

Used in: `GenericKafkaListenerConfiguration`

Full list of `GenericKafkaListenerConfigurationBootstrap` schema properties

Broker service equivalents of `nodePort`, `host`, `loadBalancerIP` and `annotations` properties are configured in the `GenericKafkaListenerConfigurationBroker` schema.

`alternativeNames`

You can specify alternative names for the bootstrap service. The names are added to the broker certificates and can be used for TLS hostname verification. The `alternativeNames` property is applicable to all types of listeners.

Example of an external `route` listener configured with an additional bootstrap address

```
listeners:
  #...
  - name: external
    port: 9094
    type: route
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        alternativeNames:
          - example.hostname1
          - example.hostname2
# ...
```

`host`

The `host` property is used with `route` and `ingress` listeners to specify the hostnames used by the bootstrap and per-broker services.

A `host` property value is mandatory for `ingress` listener configuration, as the Ingress controller does not assign any hostnames automatically. Make sure that the hostnames resolve to the Ingress endpoints. Strimzi will not perform any validation that the requested hosts are available and properly routed to the Ingress endpoints.

Example of host configuration for an ingress listener

```
listeners:
  #...
  - name: external
    port: 9094
    type: ingress
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        host: bootstrap.myingress.com
```

```
      brokers:
      - broker: 0
        host: broker-0.myingress.com
      - broker: 1
        host: broker-1.myingress.com
      - broker: 2
        host: broker-2.myingress.com
# ...
```

By default, `route` listener hosts are automatically assigned by OpenShift. However, you can override the assigned route hosts by specifying hosts.

Strimzi does not perform any validation that the requested hosts are available. You must ensure that they are free and can be used.

Example of host configuration for a route listener

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: route
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        host: bootstrap.myrouter.com
      brokers:
      - broker: 0
        host: broker-0.myrouter.com
      - broker: 1
        host: broker-1.myrouter.com
      - broker: 2
        host: broker-2.myrouter.com
# ...
```

`nodePort`

By default, the port numbers used for the bootstrap and broker services are automatically assigned by Kubernetes. You can override the assigned node ports for `nodeport` listeners by specifying the requested port numbers.

Strimzi does not perform any validation on the requested ports. You must ensure that they are free and available for use.

Example of an external listener configured with overrides for node ports

```
# ...
```

```
listeners:
  #...
  - name: external
    port: 9094
    type: nodeport
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        nodePort: 32100
      brokers:
      - broker: 0
        nodePort: 32000
      - broker: 1
        nodePort: 32001
      - broker: 2
        nodePort: 32002
# ...
```

`loadBalancerIP`

Use the `loadBalancerIP` property to request a specific IP address when creating a loadbalancer. Use this property when you need to use a loadbalancer with a specific IP address. The `loadBalancerIP` field is ignored if the cloud provider does not support the feature.

Example of an external listener of type `loadbalancer` with specific loadbalancer IP address requests

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        loadBalancerIP: 172.29.3.10
      brokers:
      - broker: 0
        loadBalancerIP: 172.29.3.1
      - broker: 1
        loadBalancerIP: 172.29.3.2
      - broker: 2
        loadBalancerIP: 172.29.3.3
# ...
```

`annotations`

Use the `annotations` property to add annotations to Kubernetes resources related to the listeners. You can use these annotations, for example, to instrument DNS tooling such as External DNS, which automatically assigns DNS names to the loadbalancer services.

Example of an external listener of type `loadbalancer` using `annotations`

```
# ...
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      bootstrap:
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-bootstrap.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
      brokers:
      - broker: 0
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-broker-0.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
      - broker: 1
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-broker-1.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
      - broker: 2
        annotations:
          external-dns.alpha.kubernetes.io/hostname: kafka-broker-2.mydomain.com.
          external-dns.alpha.kubernetes.io/ttl: "60"
# ...
```

`GenericKafkaListenerConfigurationBootstrap` schema properties

Property	Description
alternativeNames string array	Additional alternative names for the bootstrap service. The alternative names will be added to the list of subject alternative names of the TLS certificates.
host string	The bootstrap host. This field will be used in the Ingress resource or in the Route resource to specify the desired hostname. This field can be used only with `route` (optional) or `ingress` (required) type listeners.

Property	Description
nodePort integer	Node port for the bootstrap service. This field can be used only with `nodeport` type listener.
loadBalancerIP string	The loadbalancer is requested with the IP address specified in this field. This feature depends on whether the underlying cloud provider supports specifying the `loadBalancerIP` when a load balancer is created. This field is ignored if the cloud provider does not support the feature.This field can be used only with `loadbalancer` type listener.
annotations map	Annotations that will be added to the `Ingress`, `Route`, or `Service` resource. You can use this field to configure DNS providers such as External DNS. This field can be used only with `loadbalancer`, `nodeport`, `route`, or `ingress` type listeners.
labels map	Labels that will be added to the `Ingress`, `Route`, or `Service` resource. This field can be used only with `loadbalancer`, `nodeport`, `route`, or `ingress` type listeners.

4.2.14. `GenericKafkaListenerConfigurationBroker` schema reference

Used in: `GenericKafkaListenerConfiguration`

Full list of `GenericKafkaListenerConfigurationBroker` schema properties

You can see example configuration for the `nodePort`, `host`, `loadBalancerIP` and `annotations` properties in the `GenericKafkaListenerConfigurationBootstrap` schema, which configures bootstrap service overrides.

Advertised addresses for brokers

By default, Strimzi tries to automatically determine the hostnames and ports that your Kafka cluster advertises to its clients. This is not sufficient in all situations, because the infrastructure on which Strimzi is running might not provide the right hostname or port through which Kafka can be accessed.

You can specify a broker ID and customize the advertised hostname and port in the `configuration` property of the listener. Strimzi will then automatically configure the advertised address in the Kafka brokers and add it to the broker certificates so it can be used for TLS hostname verification. Overriding the advertised host and ports is available for all types of listeners.

Example of an external `route` listener configured with overrides for advertised addresses

```
listeners:
  #...
```

```
  - name: external
    port: 9094
    type: route
    tls: true
    authentication:
      type: tls
    configuration:
      brokers:
      - broker: 0
        advertisedHost: example.hostname.0
        advertisedPort: 12340
      - broker: 1
        advertisedHost: example.hostname.1
        advertisedPort: 12341
      - broker: 2
        advertisedHost: example.hostname.2
        advertisedPort: 12342
# ...
```

`GenericKafkaListenerConfigurationBroker` **schema properties**

Property	Description
broker integer	ID of the kafka broker (broker identifier). Broker IDs start from 0 and correspond to the number of broker replicas.
advertisedHost string	The host name which will be used in the brokers' `advertised.brokers`.
advertisedPort integer	The port number which will be used in the brokers' `advertised.brokers`.
host string	The broker host. This field will be used in the Ingress resource or in the Route resource to specify the desired hostname. This field can be used only with `route` (optional) or `ingress` (required) type listeners.
nodePort integer	Node port for the per-broker service. This field can be used only with `nodeport` type listener.
loadBalancerIP string	The loadbalancer is requested with the IP address specified in this field. This feature depends on whether the underlying cloud provider supports specifying the `loadBalancerIP` when a load balancer is created. This field is ignored if the cloud provider does not support the feature.This field can be used only with `loadbalancer` type listener.

Property	Description
annotations	Annotations that will be added to the `Ingress` or `Service` resource. You can use this field to configure DNS providers such as External DNS. This field can be used only with `loadbalancer`, `nodeport`, or `ingress` type listeners.
map	
labels	Labels that will be added to the `Ingress`, `Route`, or `Service` resource. This field can be used only with `loadbalancer`, `nodeport`, `route`, or `ingress` type listeners.
map	

4.2.15. `EphemeralStorage` schema reference

Used in: `JbodStorage`, `KafkaClusterSpec`, `ZookeeperClusterSpec`

The `type` property is a discriminator that distinguishes use of the `EphemeralStorage` type from `PersistentClaimStorage`. It must have the value `ephemeral` for the type `EphemeralStorage`.

Property	Description
id	Storage identification number. It is mandatory only for storage volumes defined in a storage of type 'jbod'.
integer	
sizeLimit	When type=ephemeral, defines the total amount of local storage required for this EmptyDir volume (for example 1Gi).
string	
type	Must be `ephemeral`.
string	

4.2.16. `PersistentClaimStorage` schema reference

Used in: `JbodStorage`, `KafkaClusterSpec`, `ZookeeperClusterSpec`

The `type` property is a discriminator that distinguishes use of the `PersistentClaimStorage` type from `EphemeralStorage`. It must have the value `persistent-claim` for the type `PersistentClaimStorage`.

Property	Description
type	Must be `persistent-claim`.
string	
size	When type=persistent-claim, defines the size of the persistent volume claim (i.e 1Gi). Mandatory when type=persistent-claim.
string	
selector	Specifies a specific persistent volume to use. It contains key:value pairs representing labels for selecting such a volume.
map	

Property	Description
deleteClaim boolean	Specifies if the persistent volume claim has to be deleted when the cluster is un-deployed.
class string	The storage class to use for dynamic volume allocation.
id integer	Storage identification number. It is mandatory only for storage volumes defined in a storage of type 'jbod'.
overrides `PersistentClaimStorageOverride` array	Overrides for individual brokers. The `overrides` field allows to specify a different configuration for different brokers.

4.2.17. `PersistentClaimStorageOverride` schema reference

Used in: `PersistentClaimStorage`

Property	Description
class string	The storage class to use for dynamic volume allocation for this broker.
broker integer	Id of the kafka broker (broker identifier).

4.2.18. `JbodStorage` schema reference

Used in: `KafkaClusterSpec`

The `type` property is a discriminator that distinguishes use of the `JbodStorage` type from `EphemeralStorage`, `PersistentClaimStorage`. It must have the value `jbod` for the type `JbodStorage`.

Property	Description
type string	Must be `jbod`.
volumes `EphemeralStorage`, `PersistentClaimStorage` array	List of volumes as Storage objects representing the JBOD disks array.

4.2.19. `KafkaAuthorizationSimple` schema reference

Used in: `KafkaClusterSpec`

Full list of `KafkaAuthorizationSimple` schema properties

Simple authorization in Strimzi uses the `AclAuthorizer` plugin, the default Access Control Lists (ACLs) authorization plugin provided with Apache Kafka. ACLs allow you to define which users

have access to which resources at a granular level.

Configure the `Kafka` custom resource to use simple authorization. Set the `type` property in the `authorization` section to the value `simple`, and configure a list of super users.

Access rules are configured for the `KafkaUser`, as described in the ACLRule schema reference.

`superUsers`

A list of user principals treated as super users, so that they are always allowed without querying ACL rules.

An example of simple authorization configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: simple
      superUsers:
        - CN=client_1
        - user_2
        - CN=client_3
    # ...
```

NOTE The `super.user` configuration option in the `config` property in `Kafka.spec.kafka` is ignored. Designate super users in the `authorization` property instead. For more information, see Kafka broker configuration.

`KafkaAuthorizationSimple` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaAuthorizationSimple` type from `KafkaAuthorizationOpa`, `KafkaAuthorizationKeycloak`, `KafkaAuthorizationCustom`. It must have the value `simple` for the type `KafkaAuthorizationSimple`.

Property	Description
type string	Must be `simple`.
superUsers string array	List of super users. Should contain list of user principals which should get unlimited access rights.

4.2.20. KafkaAuthorizationOpa schema reference

Used in: KafkaClusterSpec

Full list of KafkaAuthorizationOpa schema properties

To use Open Policy Agent authorization, set the type property in the authorization section to the value opa, and configure OPA properties as required. Strimzi uses Open Policy Agent plugin for Kafka authorization as the authorizer. For more information about the format of the input data and policy examples, see Open Policy Agent plugin for Kafka authorization.

url

The URL used to connect to the Open Policy Agent server. The URL has to include the policy which will be queried by the authorizer. **Required.**

allowOnError

Defines whether a Kafka client should be allowed or denied by default when the authorizer fails to query the Open Policy Agent, for example, when it is temporarily unavailable. Defaults to false - all actions will be denied.

initialCacheCapacity

Initial capacity of the local cache used by the authorizer to avoid querying the Open Policy Agent for every request. Defaults to 5000.

maximumCacheSize

Maximum capacity of the local cache used by the authorizer to avoid querying the Open Policy Agent for every request. Defaults to 50000.

expireAfterMs

The expiration of the records kept in the local cache to avoid querying the Open Policy Agent for every request. Defines how often the cached authorization decisions are reloaded from the Open Policy Agent server. In milliseconds. Defaults to 3600000 milliseconds (1 hour).

tlsTrustedCertificates

Trusted certificates for TLS connection to the OPA server.

superUsers

A list of user principals treated as super users, so that they are always allowed without querying the open Policy Agent policy.

An example of Open Policy Agent authorizer configuration

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

```
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: opa
      url: http://opa:8181/v1/data/kafka/allow
      allowOnError: false
      initialCacheCapacity: 1000
      maximumCacheSize: 10000
      expireAfterMs: 60000
      superUsers:
        - CN=fred
        - sam
        - CN=edward
    # ...
```

`KafkaAuthorizationOpa` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaAuthorizationOpa` type from `KafkaAuthorizationSimple`, `KafkaAuthorizationKeycloak`, `KafkaAuthorizationCustom`. It must have the value `opa` for the type `KafkaAuthorizationOpa`.

Property	Description
type	Must be `opa`.
string	
url	The URL used to connect to the Open Policy Agent server. The URL has to include the policy which will be queried by the authorizer. This option is required.
string	
allowOnError	Defines whether a Kafka client should be allowed or denied by default when the authorizer fails to query the Open Policy Agent, for example, when it is temporarily unavailable). Defaults to `false` - all actions will be denied.
boolean	
initialCacheCapacity	Initial capacity of the local cache used by the authorizer to avoid querying the Open Policy Agent for every request Defaults to `5000`.
integer	
maximumCacheSize	Maximum capacity of the local cache used by the authorizer to avoid querying the Open Policy Agent for every request. Defaults to `50000`.
integer	

Property	Description
expireAfterMs	The expiration of the records kept in the local cache to avoid querying the Open Policy Agent for every request. Defines how often the cached authorization decisions are reloaded from the Open Policy Agent server. In milliseconds. Defaults to `3600000`.
integer	
tlsTrustedCertificates	Trusted certificates for TLS connection to the OPA server.
`CertSecretSource` array	
superUsers	List of super users, which is specifically a list of user principals that have unlimited access rights.
string array	
enableMetrics	Defines whether the Open Policy Agent authorizer plugin should provide metrics. Defaults to `false`.
boolean	

4.2.21. `KafkaAuthorizationKeycloak` schema reference

Used in: `KafkaClusterSpec`

The `type` property is a discriminator that distinguishes use of the `KafkaAuthorizationKeycloak` type from `KafkaAuthorizationSimple`, `KafkaAuthorizationOpa`, `KafkaAuthorizationCustom`. It must have the value `keycloak` for the type `KafkaAuthorizationKeycloak`.

Property	Description
type	Must be `keycloak`.
string	
clientId	OAuth Client ID which the Kafka client can use to authenticate against the OAuth server and use the token endpoint URI.
string	
tokenEndpointUri	Authorization server token endpoint URI.
string	
tlsTrustedCertificates	Trusted certificates for TLS connection to the OAuth server.
`CertSecretSource` array	
disableTlsHostnameVerification	Enable or disable TLS hostname verification. Default value is `false`.
boolean	
delegateToKafkaAcls	Whether authorization decision should be delegated to the 'Simple' authorizer if DENIED by Keycloak Authorization Services policies. Default value is `false`.
boolean	

Property	Description
grantsRefreshPeriodSeconds	The time between two consecutive grants refresh runs in seconds. The default value is 60.
integer	
grantsRefreshPoolSize	The number of threads to use to refresh grants for active sessions. The more threads, the more parallelism, so the sooner the job completes. However, using more threads places a heavier load on the authorization server. The default value is 5.
integer	
superUsers	List of super users. Should contain list of user principals which should get unlimited access rights.
string array	
connectTimeoutSeconds	The connect timeout in seconds when connecting to authorization server. If not set, the effective connect timeout is 60 seconds.
integer	
readTimeoutSeconds	The read timeout in seconds when connecting to authorization server. If not set, the effective read timeout is 60 seconds.
integer	
httpRetries	The maximum number of retries to attempt if an initial HTTP request fails. If not set, the default is to not attempt any retries.
integer	
enableMetrics	Enable or disable OAuth metrics. Default value is `false`.
boolean	

4.2.22. `KafkaAuthorizationCustom` schema reference

Used in: `KafkaClusterSpec`

Full list of `KafkaAuthorizationCustom` schema properties

To use custom authorization in Strimzi, you can configure your own `Authorizer` plugin to define Access Control Lists (ACLs).

ACLs allow you to define which users have access to which resources at a granular level.

Configure the `Kafka` custom resource to use custom authorization. Set the `type` property in the `authorization` section to the value `custom`, and the set following properties.

IMPORTANT	The custom authorizer must implement the `org.apache.kafka.server.authorizer.Authorizer` interface, and support configuration of `super.users` using the super.users configuration property.

`authorizerClass`

(Required) Java class that implements the `org.apache.kafka.server.authorizer.Authorizer` interface

to support custom ACLs.

`superUsers`

A list of user principals treated as super users, so that they are always allowed without querying ACL rules.

You can add configuration for initializing the custom authorizer using `Kafka.spec.kafka.config`.

An example of custom authorization configuration under `Kafka.spec`

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: myproject
spec:
  kafka:
    # ...
    authorization:
      type: custom
      authorizerClass: io.mycompany.CustomAuthorizer
      superUsers:
        - CN=client_1
        - user_2
        - CN=client_3
    # ...
    config:
      authorization.custom.property1=value1
      authorization.custom.property2=value2
    # ...
```

In addition to the `Kafka` custom resource configuration, the JAR file containing the custom authorizer class along with its dependencies must be available on the classpath of the Kafka broker.

The Strimzi Maven build process provides a mechanism to add custom third-party libraries to the generated Kafka broker container image by adding them as dependencies in the `pom.xml` file under the `docker-images/kafka/kafka-thirdparty-libs` directory. The directory contains different folders for different Kafka versions. Choose the appropriate folder. Before modifying the `pom.xml` file, the third-party library must be available in a Maven repository, and that Maven repository must be accessible to the Strimzi build process.

NOTE The `super.user` configuration option in the `config` property in `Kafka.spec.kafka` is ignored. Designate super users in the `authorization` property instead. For more information, see Kafka broker configuration.

Custom authorization can make use of group membership information extracted from the JWT token during authentication when using `oauth` authentication and configuring `groupsClaim` configuration attribute. Groups are available on the `OAuthKafkaPrincipal` object during authorize() call as follows:

```
    public List<AuthorizationResult> authorize(AuthorizableRequestContext
requestContext, List<Action> actions) {

        KafkaPrincipal principal = requestContext.principal();
        if (principal instanceof OAuthKafkaPrincipal) {
            OAuthKafkaPrincipal p = (OAuthKafkaPrincipal) principal;

            for (String group: p.getGroups()) {
                System.out.println("Group: " + group);
            }
        }
    }
```

`KafkaAuthorizationCustom` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaAuthorizationCustom` type from `KafkaAuthorizationSimple`, `KafkaAuthorizationOpa`, `KafkaAuthorizationKeycloak`. It must have the value `custom` for the type `KafkaAuthorizationCustom`.

Property	Description
type string	Must be `custom`.
authorizerClass string	Authorization implementation class, which must be available in classpath.
superUsers string array	List of super users, which are user principals with unlimited access rights.
supportsAdminApi boolean	Indicates whether the custom authorizer supports the APIs for managing ACLs using the Kafka Admin API. Defaults to `false`.

4.2.23. `Rack` schema reference

Used in: `KafkaBridgeSpec`, `KafkaClusterSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`

Full list of `Rack` schema properties

The `rack` option configures rack awareness. A *rack* can represent an availability zone, data center, or an actual rack in your data center. The *rack* is configured through a `topologyKey`. `topologyKey` identifies a label on Kubernetes nodes that contains the name of the topology in its value. An example of such a label is `topology.kubernetes.io/zone` (or `failure-domain.beta.kubernetes.io/zone` on older Kubernetes versions), which contains the name of the availability zone in which the Kubernetes node runs. You can configure your Kafka cluster to be aware of the *rack* in which it runs, and enable additional features such as spreading partition replicas across different racks or consuming messages from the closest replicas.

For more information about Kubernetes node labels, see Well-Known Labels, Annotations and Taints. Consult your Kubernetes administrator regarding the node label that represents the zone or rack into which the node is deployed.

Spreading partition replicas across racks

When rack awareness is configured, Strimzi will set `broker.rack` configuration for each Kafka broker. The `broker.rack` configuration assigns a rack ID to each broker. When `broker.rack` is configured, Kafka brokers will spread partition replicas across as many different racks as possible. When replicas are spread across multiple racks, the probability that multiple replicas will fail at the same time is lower than if they would be in the same rack. Spreading replicas improves resiliency, and is important for availability and reliability. To enable rack awareness in Kafka, add the `rack` option to the `.spec.kafka` section of the `Kafka` custom resource as shown in the example below.

Example `rack` configuration for Kafka

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    # ...
```

NOTE The *rack* in which brokers are running can change in some cases when the pods are deleted or restarted. As a result, the replicas running in different racks might then share the same rack. Use Cruise Control and the `KafkaRebalance` resource with the `RackAwareGoal` to make sure that replicas remain distributed across different racks.

When rack awareness is enabled in the `Kafka` custom resource, Strimzi will automatically add the Kubernetes `preferredDuringSchedulingIgnoredDuringExecution` affinity rule to distribute the Kafka brokers across the different racks. However, the *preferred* rule does not guarantee that the brokers will be spread. Depending on your exact Kubernetes and Kafka configurations, you should add additional `affinity` rules or configure `topologySpreadConstraints` for both ZooKeeper and Kafka to make sure the nodes are properly distributed accross as many racks as possible. For more information see Configuring pod scheduling.

Consuming messages from the closest replicas

Rack awareness can also be used in consumers to fetch data from the closest replica. This is useful for reducing the load on your network when a Kafka cluster spans multiple datacenters and can also reduce costs when running Kafka in public clouds. However, it can lead to increased latency.

In order to be able to consume from the closest replica, rack awareness has to be configured in the Kafka cluster, and the `RackAwareReplicaSelector` has to be enabled. The replica selector plugin provides the logic that enables clients to consume from the nearest replica. The default

implementation uses `LeaderSelector` to always select the leader replica for the client. Specify `RackAwareReplicaSelector` for the `replica.selector.class` to switch from the default implementation.

Example `rack` configuration with enabled replica-aware selector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    rack:
      topologyKey: topology.kubernetes.io/zone
    config:
      # ...
      replica.selector.class: org.apache.kafka.common.replica.RackAwareReplicaSelector
    # ...
```

In addition to the Kafka broker configuration, you also need to specify the `client.rack` option in your consumers. The `client.rack` option should specify the *rack ID* in which the consumer is running. `RackAwareReplicaSelector` associates matching `broker.rack` and `client.rack` IDs, to find the nearest replica and consume from it. If there are multiple replicas in the same rack, `RackAwareReplicaSelector` always selects the most up-to-date replica. If the rack ID is not specified, or if it cannot find a replica with the same rack ID, it will fall back to the leader replica.

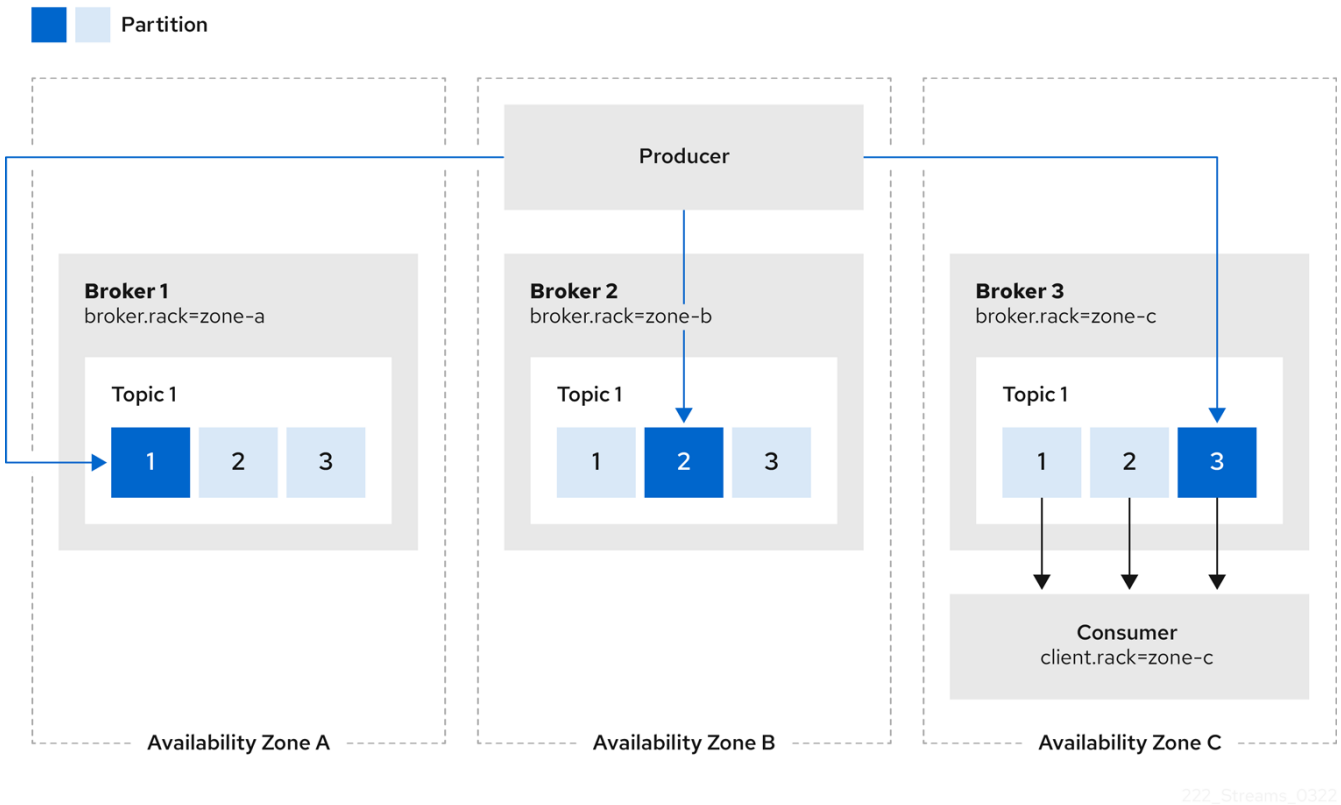


Figure 3. Example showing client consuming from replicas in the same availability zone

You can also configure Kafka Connect, MirrorMaker 2 and Kafka Bridge so that connectors consume messages from the closest replicas. You enable rack awareness in the `KafkaConnect`, `KafkaMirrorMaker2`, and `KafkaBridge` custom resources. The configuration does does not set affinity rules, but you can also configure `affinity` or `topologySpreadConstraints`. For more information see Configuring pod scheduling.

When deploying Kafka Connect using Strimzi, you can use the `rack` section in the `KafkaConnect` custom resource to automatically configure the `client.rack` option.

Example `rack` configuration for Kafka Connect

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

When deploying MirrorMaker 2 using Strimzi, you can use the `rack` section in the `KafkaMirrorMaker2` custom resource to automatically configure the `client.rack` option.

Example `rack` configuration for MirrorMaker 2

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
# ...
spec:
  # ...
  rack:
    topologyKey: topology.kubernetes.io/zone
  # ...
```

When deploying Kafka Bridge using Strimzi, you can use the `rack` section in the `KafkaBridge` custom resource to automatically configure the `client.rack` option.

Example `rack` configuration for Kafka Bridge

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

`Rack` schema properties

Property	Description
topologyKey	A key that matches labels assigned to the Kubernetes cluster nodes. The value of the label is used to set a broker's `broker.rack` config, and the `client.rack` config for Kafka Connect or MirrorMaker 2.
string	

4.2.24. `Probe` schema reference

Used in: `CruiseControlSpec`, `EntityTopicOperatorSpec`, `EntityUserOperatorSpec`, `KafkaBridgeSpec`, `KafkaClusterSpec`, `KafkaConnectSpec`, `KafkaExporterSpec`, `KafkaMirrorMaker2Spec`, `KafkaMirrorMakerSpec`, `TlsSidecar`, `ZookeeperClusterSpec`

Property	Description
failureThreshold	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 3. Minimum value is 1.
integer	
initialDelaySeconds	The initial delay before first the health is first checked. Default to 15 seconds. Minimum value is 0.
integer	
periodSeconds	How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.
integer	
successThreshold	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1. Must be 1 for liveness. Minimum value is 1.
integer	
timeoutSeconds	The timeout for each attempted health check. Default to 5 seconds. Minimum value is 1.
integer	

4.2.25. `JvmOptions` schema reference

Used in: `CruiseControlSpec`, `EntityTopicOperatorSpec`, `EntityUserOperatorSpec`, `KafkaBridgeSpec`, `KafkaClusterSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`, `KafkaMirrorMakerSpec`, `ZookeeperClusterSpec`

Property	Description
-XX	A map of -XX options to the JVM.
map	
-Xms	-Xms option to to the JVM.
string	

Property	Description
-Xmx	-Xmx option to to the JVM.
string	
gcLoggingEnabled	Specifies whether the Garbage Collection logging is enabled. The default is false.
boolean	
javaSystemProperties	A map of additional system properties which will be passed using the `-D` option to the JVM.
`SystemProperty` array	

4.2.26. `SystemProperty` schema reference

Used in: `JvmOptions`

Property	Description
name	The system property name.
string	
value	The system property value.
string	

4.2.27. `KafkaJmxOptions` schema reference

Used in: `KafkaClusterSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`, `ZookeeperClusterSpec`

Full list of `KafkaJmxOptions` schema properties

Configures JMX connection options.

Get JMX metrics from Kafka brokers, ZooKeeper nodes, Kafka Connect, and MirrorMaker 2. by connecting to port 9999. Use the `jmxOptions` property to configure a password-protected or an unprotected JMX port. Using password protection prevents unauthorized pods from accessing the port.

You can then obtain metrics about the component.

For example, for each Kafka broker you can obtain bytes-per-second usage data from clients, or the request rate of the network of the broker.

To enable security for the JMX port, set the `type` parameter in the `authentication` field to `password`.

Example password-protected JMX configuration for Kafka brokers and ZooKeeper nodes

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

```
    # ...
    jmxOptions:
      authentication:
        type: "password"
    # ...
  zookeeper:
    # ...
    jmxOptions:
      authentication:
        type: "password"
    #...
```

You can then deploy a pod into a cluster and obtain JMX metrics using the headless service by specifying which broker you want to address.

For example, to get JMX metrics from broker *0* you specify:

```
"CLUSTER-NAME-kafka-0.CLUSTER-NAME-kafka-brokers"
```

`CLUSTER-NAME-kafka-0` is name of the broker pod, and `CLUSTER-NAME-kafka-brokers` is the name of the headless service to return the IPs of the broker pods.

If the JMX port is secured, you can get the username and password by referencing them from the JMX Secret in the deployment of your pod.

For an unprotected JMX port, use an empty object `{}` to open the JMX port on the headless service. You deploy a pod and obtain metrics in the same way as for the protected port, but in this case any pod can read from the JMX port.

Example open port JMX configuration for Kafka brokers and ZooKeeper nodes

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    jmxOptions: {}
    # ...
  zookeeper:
    # ...
    jmxOptions: {}
    # ...
```

Additional resources

- For more information on the Kafka component metrics exposed using JMX, see the Apache Kafka documentation.

KafkaJmxOptions schema properties

Property	Description
authentication	Authentication configuration for connecting to the JMX port. The type depends on the value of the authentication.type property within the given object, which must be one of [password].
KafkaJmxAuthenticationPassword	

4.2.28. KafkaJmxAuthenticationPassword schema reference

Used in: KafkaJmxOptions

The type property is a discriminator that distinguishes use of the KafkaJmxAuthenticationPassword type from other subtypes which may be added in the future. It must have the value password for the type KafkaJmxAuthenticationPassword.

Property	Description
type	Must be password.
string	

4.2.29. JmxPrometheusExporterMetrics schema reference

Used in: CruiseControlSpec, KafkaClusterSpec, KafkaConnectSpec, KafkaMirrorMaker2Spec, KafkaMirrorMakerSpec, ZookeeperClusterSpec

The type property is a discriminator that distinguishes use of the JmxPrometheusExporterMetrics type from other subtypes which may be added in the future. It must have the value jmxPrometheusExporter for the type JmxPrometheusExporterMetrics.

Property	Description
type	Must be jmxPrometheusExporter.
string	
valueFrom	ConfigMap entry where the Prometheus JMX Exporter configuration is stored. For details of the structure of this configuration, see the Prometheus JMX Exporter.
ExternalConfigurationReference	

4.2.30. ExternalConfigurationReference schema reference

Used in: ExternalLogging, JmxPrometheusExporterMetrics

Property	Description
configMapKeyRef	Reference to the key in the ConfigMap containing the configuration. For more information, see the external documentation for core/v1 configmapkeyselector.
ConfigMapKeySelector	

4.2.31. `InlineLogging` schema reference

Used in: `CruiseControlSpec`, `EntityTopicOperatorSpec`, `EntityUserOperatorSpec`, `KafkaBridgeSpec`, `KafkaClusterSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`, `KafkaMirrorMakerSpec`, `ZookeeperClusterSpec`

The `type` property is a discriminator that distinguishes use of the `InlineLogging` type from `ExternalLogging`. It must have the value `inline` for the type `InlineLogging`.

Property	Description
type	Must be `inline`.
string	
loggers	A Map from logger name to logger level.
map	

4.2.32. `ExternalLogging` schema reference

Used in: `CruiseControlSpec`, `EntityTopicOperatorSpec`, `EntityUserOperatorSpec`, `KafkaBridgeSpec`, `KafkaClusterSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`, `KafkaMirrorMakerSpec`, `ZookeeperClusterSpec`

The `type` property is a discriminator that distinguishes use of the `ExternalLogging` type from `InlineLogging`. It must have the value `external` for the type `ExternalLogging`.

Property	Description
type	Must be `external`.
string	
valueFrom	`ConfigMap` entry where the logging configuration is stored.
`ExternalConfigurationReference`	

4.2.33. `KafkaClusterTemplate` schema reference

Used in: `KafkaClusterSpec`

Property	Description
statefulset	**The `statefulset` property has been deprecated.** Support for StatefulSets was removed in Strimzi 0.35.0. This property is ignored. Template for Kafka `StatefulSet`.
`StatefulSetTemplate`	

Property	Description
pod `PodTemplate`	Template for Kafka `Pods`.
bootstrapService `InternalServiceTemplate`	Template for Kafka bootstrap `Service`.
brokersService `InternalServiceTemplate`	Template for Kafka broker `Service`.
externalBootstrapService `ResourceTemplate`	Template for Kafka external bootstrap `Service`.
perPodService `ResourceTemplate`	Template for Kafka per-pod `Services` used for access from outside of Kubernetes.
externalBootstrapRoute `ResourceTemplate`	Template for Kafka external bootstrap `Route`.
perPodRoute `ResourceTemplate`	Template for Kafka per-pod `Routes` used for access from outside of OpenShift.
externalBootstrapIngress `ResourceTemplate`	Template for Kafka external bootstrap `Ingress`.
perPodIngress `ResourceTemplate`	Template for Kafka per-pod `Ingress` used for access from outside of Kubernetes.
persistentVolumeClaim `ResourceTemplate`	Template for all Kafka `PersistentVolumeClaims`.
podDisruptionBudget `PodDisruptionBudgetTemplate`	Template for Kafka `PodDisruptionBudget`.
kafkaContainer `ContainerTemplate`	Template for the Kafka broker container.
initContainer `ContainerTemplate`	Template for the Kafka init container.
clusterCaCert `ResourceTemplate`	Template for Secret with Kafka Cluster certificate public key.
serviceAccount `ResourceTemplate`	Template for the Kafka service account.
jmxSecret `ResourceTemplate`	Template for Secret of the Kafka Cluster JMX authentication.
clusterRoleBinding `ResourceTemplate`	Template for the Kafka ClusterRoleBinding.
podSet `ResourceTemplate`	Template for Kafka `StrimziPodSet` resource.

4.2.34. `StatefulSetTemplate` schema reference

Used in: `KafkaClusterTemplate`, `ZookeeperClusterTemplate`

Property	Description
metadata	Metadata applied to the resource.
`MetadataTemplate`	
podManagementPolicy	PodManagementPolicy which will be used for this StatefulSet. Valid values are `Parallel` and `OrderedReady`. Defaults to `Parallel`.
string (one of [OrderedReady, Parallel])	

4.2.35. `MetadataTemplate` schema reference

Used in: `BuildConfigTemplate`, `DeploymentTemplate`, `InternalServiceTemplate`, `PodDisruptionBudgetTemplate`, `PodTemplate`, `ResourceTemplate`, `StatefulSetTemplate`

Full list of `MetadataTemplate` schema properties

`Labels` and `Annotations` are used to identify and organize resources, and are configured in the `metadata` property.

For example:

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

The `labels` and `annotations` fields can contain any labels or annotations that do not contain the reserved string `strimzi.io`. Labels and annotations containing `strimzi.io` are used internally by Strimzi and cannot be configured.

`MetadataTemplate` schema properties

Property	Description
labels	Labels added to the Kubernetes resource.
map	
annotations	Annotations added to the Kubernetes resource.
map	

4.2.36. PodTemplate schema reference

Used in: CruiseControlTemplate, EntityOperatorTemplate, JmxTransTemplate, KafkaBridgeTemplate, KafkaClusterTemplate, KafkaConnectTemplate, KafkaExporterTemplate, KafkaMirrorMakerTemplate, ZookeeperClusterTemplate

Full list of PodTemplate schema properties

Configures the template for Kafka pods.

Example PodTemplate configuration

```
# ...
template:
  pod:
    metadata:
      labels:
        label1: value1
      annotations:
        anno1: value1
    imagePullSecrets:
      - name: my-docker-credentials
    securityContext:
      runAsUser: 1000001
      fsGroup: 0
    terminationGracePeriodSeconds: 120
# ...
```

hostAliases

Use the hostAliases property to a specify a list of hosts and IP addresses, which are injected into the /etc/hosts file of the pod.

This configuration is especially useful for Kafka Connect or MirrorMaker when a connection outside of the cluster is also requested by users.

Example hostAliases configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
#...
spec:
  # ...
  template:
    pod:
      hostAliases:
      - ip: "192.168.1.86"
        hostnames:
        - "my-host-1"
        - "my-host-2"
      #...
```

`PodTemplate` schema properties

Property	Description
metadata `MetadataTemplate`	Metadata applied to the resource.
imagePullSecrets LocalObjectReference array	List of references to secrets in the same namespace to use for pulling any of the images used by this Pod. When the `STRIMZI_IMAGE_PULL_SECRETS` environment variable in Cluster Operator and the `imagePullSecrets` option are specified, only the `imagePullSecrets` variable is used and the `STRIMZI_IMAGE_PULL_SECRETS` variable is ignored. For more information, see the external documentation for core/v1 localobjectreference.
securityContext PodSecurityContext	Configures pod-level security attributes and common container settings. For more information, see the external documentation for core/v1 podsecuritycontext.
terminationGracePeriodSeconds integer	The grace period is the duration in seconds after the processes running in the pod are sent a termination signal, and the time when the processes are forcibly halted with a kill signal. Set this value to longer than the expected cleanup time for your process. Value must be a non-negative integer. A zero value indicates delete immediately. You might need to increase the grace period for very large Kafka clusters, so that the Kafka brokers have enough time to transfer their work to another broker before they are terminated. Defaults to 30 seconds.
affinity Affinity	The pod's affinity rules. For more information, see the external documentation for core/v1 affinity.
tolerations Toleration array	The pod's tolerations. For more information, see the external documentation for core/v1 toleration.
priorityClassName string	The name of the priority class used to assign priority to the pods. For more information about priority classes, see Pod Priority and Preemption.
schedulerName string	The name of the scheduler used to dispatch this `Pod`. If not specified, the default scheduler will be used.

Property	Description
hostAliases HostAlias array	The pod's HostAliases. HostAliases is an optional list of hosts and IPs that will be injected into the Pod's hosts file if specified. For more information, see the external documentation for core/v1 hostalias.
tmpDirSizeLimit string	Defines the total amount (for example `1Gi`) of local storage required for temporary EmptyDir volume (`/tmp`). Default value is `5Mi`.
enableServiceLinks boolean	Indicates whether information about services should be injected into Pod's environment variables.
topologySpreadConstraints TopologySpreadConstraint array	The pod's topology spread constraints. For more information, see the external documentation for core/v1 topologyspreadconstraint.

4.2.37. `InternalServiceTemplate` schema reference

Used in: `CruiseControlTemplate`, `KafkaBridgeTemplate`, `KafkaClusterTemplate`, `KafkaConnectTemplate`, `ZookeeperClusterTemplate`

Property	Description
metadata `MetadataTemplate`	Metadata applied to the resource.
ipFamilyPolicy string (one of [RequireDualStack, SingleStack, PreferDualStack])	Specifies the IP Family Policy used by the service. Available options are `SingleStack`, `PreferDualStack` and `RequireDualStack`. `SingleStack` is for a single IP family. `PreferDualStack` is for two IP families on dual-stack configured clusters or a single IP family on single-stack clusters. `RequireDualStack` fails unless there are two IP families on dual-stack configured clusters. If unspecified, Kubernetes will choose the default value based on the service type. Available on Kubernetes 1.20 and newer.
ipFamilies string (one or more of [IPv6, IPv4]) array	Specifies the IP Families used by the service. Available options are `IPv4` and `IPv6`. If unspecified, Kubernetes will choose the default value based on the `ipFamilyPolicy` setting. Available on Kubernetes 1.20 and newer.

4.2.38. `ResourceTemplate` schema reference

Used in: `CruiseControlTemplate`, `EntityOperatorTemplate`, `JmxTransTemplate`, `KafkaBridgeTemplate`,

KafkaClusterTemplate, KafkaConnectTemplate, KafkaExporterTemplate, KafkaMirrorMakerTemplate, KafkaUserTemplate, ZookeeperClusterTemplate

Property	Description
metadata	Metadata applied to the resource.
MetadataTemplate	

4.2.39. PodDisruptionBudgetTemplate schema reference

Used in: CruiseControlTemplate, KafkaBridgeTemplate, KafkaClusterTemplate, KafkaConnectTemplate, KafkaMirrorMakerTemplate, ZookeeperClusterTemplate

Full list of PodDisruptionBudgetTemplate schema properties

A PodDisruptionBudget (PDB) is a Kubernetes resource that ensures high availability by specifying the minimum number of pods that must be available during planned maintenance or upgrades. Strimzi creates a PDB for every new StrimziPodSet or Deployment. By default, the PDB allows only one pod to be unavailable at any given time. You can increase the number of unavailable pods allowed by changing the default value of the maxUnavailable property.

StrimziPodSet custom resources manage pods using a custom controller that cannot use the maxUnavailable value directly. Instead, the maxUnavailable value is automatically converted to a minAvailable value when creating the PDB resource, which effectively serves the same purpose, as illustrated in the following examples:

- If there are three broker pods and the maxUnavailable property is set to 1 in the Kafka resource, the minAvailable setting is 2, allowing one pod to be unavailable.
- If there are three broker pods and the maxUnavailable property is set to 0 (zero), the minAvailable setting is 3, requiring all three broker pods to be available and allowing zero pods to be unavailable.

Example PodDisruptionBudget template configuration

```
# ...
template:
  podDisruptionBudget:
    metadata:
      labels:
        key1: label1
        key2: label2
      annotations:
        key1: label1
        key2: label2
    maxUnavailable: 1
# ...
```

`PodDisruptionBudgetTemplate` **schema properties**

Property	Description
metadata `MetadataTemplate`	Metadata to apply to the `PodDisruptionBudgetTemplate` resource.
maxUnavailable integer	Maximum number of unavailable pods to allow automatic Pod eviction. A Pod eviction is allowed when the `maxUnavailable` number of pods or fewer are unavailable after the eviction. Setting this value to 0 prevents all voluntary evictions, so the pods must be evicted manually. Defaults to 1.

4.2.40. `ContainerTemplate` schema reference

Used in: `CruiseControlTemplate`, `EntityOperatorTemplate`, `JmxTransTemplate`, `KafkaBridgeTemplate`, `KafkaClusterTemplate`, `KafkaConnectTemplate`, `KafkaExporterTemplate`, `KafkaMirrorMakerTemplate`, `ZookeeperClusterTemplate`

Full list of `ContainerTemplate` schema properties

You can set custom security context and environment variables for a container.

The environment variables are defined under the `env` property as a list of objects with `name` and `value` fields. The following example shows two custom environment variables and a custom security context set for the Kafka broker containers:

```
# ...
template:
  kafkaContainer:
    env:
    - name: EXAMPLE_ENV_1
      value: example.env.one
    - name: EXAMPLE_ENV_2
      value: example.env.two
    securityContext:
      runAsUser: 2000
# ...
```

Environment variables prefixed with `KAFKA_` are internal to Strimzi and should be avoided. If you set a custom environment variable that is already in use by Strimzi, it is ignored and a warning is recorded in the log.

`ContainerTemplate` **schema properties**

Property	Description
env `ContainerEnvVar` array	Environment variables which should be applied to the container.
securityContext SecurityContext	Security context for the container. For more information, see the external documentation for core/v1 securitycontext.

4.2.41. `ContainerEnvVar` schema reference

Used in: `ContainerTemplate`

Property	Description
name string	The environment variable key.
value string	The environment variable value.

4.2.42. `ZookeeperClusterSpec` schema reference

Used in: `KafkaSpec`

Full list of `ZookeeperClusterSpec` schema properties

Configures a ZooKeeper cluster.

`config`

Use the `config` properties to configure ZooKeeper options as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the ZooKeeper documentation.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Security (encryption, authentication, and authorization)
- Listener configuration
- Configuration of data directories

- ZooKeeper cluster composition

Properties with the following prefixes cannot be set:

- `4lw.commands.whitelist`
- `authProvider`
- `clientPort`
- `dataDir`
- `dataLogDir`
- `quorum.auth`
- `reconfigEnabled`
- `requireClientAuthScheme`
- `secureClientPort`
- `server.`
- `snapshot.trust.empty`
- `standaloneEnabled`
- `serverCnxnFactory`
- `ssl.`
- `sslQuorum`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to ZooKeeper, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites

Example ZooKeeper configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  kafka:
    # ...
  zookeeper:
    # ...
    config:
      autopurge.snapRetainCount: 3
      autopurge.purgeInterval: 2
    # ...
```

`logging`

ZooKeeper has a configurable logger:

- `zookeeper.root.logger`

ZooKeeper uses the Apache `log4j` logger implementation.

Use the `logging` property to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. Inside the ConfigMap, the logging configuration is described using `log4j.properties`. Both `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. For more information about log levels, see Apache logging services.

Here we see examples of `inline` and `external` logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  zookeeper:
    # ...
    logging:
      type: inline
      loggers:
        zookeeper.root.logger: "INFO"
    # ...
```

External logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
spec:
  # ...
  zookeeper:
    # ...
    logging:
      type: external
      valueFrom:
        configMapKeyRef:
          name: customConfigMap
          key: zookeeper-log4j.properties
  # ...
```

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`ZookeeperClusterSpec` schema properties

Property	Description
replicas	The number of pods in the cluster.
integer	
image	The docker image for the pods.
string	
storage	Storage configuration (disk). Cannot be updated. The type depends on the value of the `storage.type` property within the given object, which must be one of [ephemeral, persistent-claim].
`EphemeralStorage`, `PersistentClaimStorage`	
config	The ZooKeeper broker config. Properties with the following prefixes cannot be set: server., dataDir, dataLogDir, clientPort, authProvider, quorum.auth, requireClientAuthScheme, snapshot.trust.empty, standaloneEnabled, reconfigEnabled, 4lw.commands.whitelist, secureClientPort, ssl., serverCnxnFactory, sslQuorum (with the exception of: ssl.protocol, ssl.quorum.protocol, ssl.enabledProtocols, ssl.quorum.enabledProtocols, ssl.ciphersuites, ssl.quorum.ciphersuites, ssl.hostnameVerification, ssl.quorum.hostnameVerification).
map	
livenessProbe	Pod liveness checking.
`Probe`	
readinessProbe	Pod readiness checking.
`Probe`	
jvmOptions	JVM Options for pods.
`JvmOptions`	
jmxOptions	JMX Options for Zookeeper nodes.
`KafkaJmxOptions`	
resources	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
metricsConfig	Metrics configuration. The type depends on the value of the `metricsConfig.type` property within the given object, which must be one of [jmxPrometheusExporter].
`JmxPrometheusExporterMetrics`	

Property	Description
logging `InlineLogging`, `ExternalLogging`	Logging configuration for ZooKeeper. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
template `ZookeeperClusterTemplate`	Template for ZooKeeper cluster resources. The template allows users to specify how the Kubernetes resources are generated.

4.2.43. `ZookeeperClusterTemplate` schema reference

Used in: `ZookeeperClusterSpec`

Property	Description
statefulset `StatefulSetTemplate`	**The `statefulset` property has been deprecated.** Support for StatefulSets was removed in Strimzi 0.35.0. This property is ignored. Template for ZooKeeper `StatefulSet`.
pod `PodTemplate`	Template for ZooKeeper `Pods`.
clientService `InternalServiceTemplate`	Template for ZooKeeper client `Service`.
nodesService `InternalServiceTemplate`	Template for ZooKeeper nodes `Service`.
persistentVolumeClaim `ResourceTemplate`	Template for all ZooKeeper `PersistentVolumeClaims`.
podDisruptionBudget `PodDisruptionBudgetTemplate`	Template for ZooKeeper `PodDisruptionBudget`.
zookeeperContainer `ContainerTemplate`	Template for the ZooKeeper container.
serviceAccount `ResourceTemplate`	Template for the ZooKeeper service account.
jmxSecret `ResourceTemplate`	Template for Secret of the Zookeeper Cluster JMX authentication.
podSet `ResourceTemplate`	Template for ZooKeeper `StrimziPodSet` resource.

4.2.44. `EntityOperatorSpec` schema reference

Used in: `KafkaSpec`

Property	Description
topicOperator	Configuration of the Topic Operator.
EntityTopicOperatorSpec	
userOperator	Configuration of the User Operator.
EntityUserOperatorSpec	
tlsSidecar	TLS sidecar configuration.
TlsSidecar	
template	Template for Entity Operator resources. The template allows users to specify how a Deployment and Pod is generated.
EntityOperatorTemplate	

4.2.45. EntityTopicOperatorSpec schema reference

Used in: EntityOperatorSpec

Full list of EntityTopicOperatorSpec schema properties

Configures the Topic Operator.

logging

The Topic Operator has a configurable logger:

- rootLogger.level

The Topic Operator uses the Apache log4j2 logger implementation.

Use the logging property in the entityOperator.topicOperator field of the Kafka resource Kafka resource to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set logging.valueFrom.configMapKeyRef.name property to the name of the ConfigMap containing the external logging configuration. Inside the ConfigMap, the logging configuration is described using log4j2.properties. Both logging.valueFrom.configMapKeyRef.name and logging.valueFrom.configMapKeyRef.key properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. For more information about log levels, see Apache logging services.

Here we see examples of inline and external logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
```

```
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
  # ...
```

External logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    topicOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: external
        valueFrom:
          configMapKeyRef:
            name: customConfigMap
            key: topic-operator-log4j2.properties
  # ...
```

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`EntityTopicOperatorSpec` schema properties

Property	Description
watchedNamespace	The namespace the Topic Operator should watch.
string	
image	The image to use for the Topic Operator.
string	
reconciliationIntervalSeconds	Interval between periodic reconciliations.
integer	
zookeeperSessionTimeoutSeconds	Timeout for the ZooKeeper session.
integer	
startupProbe	Pod startup checking.
`Probe`	
livenessProbe	Pod liveness checking.
`Probe`	
readinessProbe	Pod readiness checking.
`Probe`	
resources	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
topicMetadataMaxAttempts	The number of attempts at getting topic metadata.
integer	
logging	Logging configuration. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
`InlineLogging`, `ExternalLogging`	
jvmOptions	JVM Options for pods.
`JvmOptions`	

4.2.46. `EntityUserOperatorSpec` schema reference

Used in: `EntityOperatorSpec`

Full list of `EntityUserOperatorSpec` schema properties

Configures the User Operator.

`logging`

The User Operator has a configurable logger:

- `rootLogger.level`

The User Operator uses the Apache `log4j2` logger implementation.

Use the `logging` property in the `entityOperator.userOperator` field of the `Kafka` resource to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. Inside the ConfigMap, the logging configuration is described using `log4j2.properties`. Both `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. For more information about log levels, see Apache logging services.

Here we see examples of `inline` and `external` logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: inline
        loggers:
          rootLogger.level: INFO
  # ...
```

External logging

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

```
    # ...
    userOperator:
      watchedNamespace: my-topic-namespace
      reconciliationIntervalSeconds: 60
      logging:
        type: external
        valueFrom:
          configMapKeyRef:
            name: customConfigMap
            key: user-operator-log4j2.properties
  # ...
```

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`EntityUserOperatorSpec` schema properties

Property	Description
watchedNamespace	The namespace the User Operator should watch.
string	
image	The image to use for the User Operator.
string	
reconciliationIntervalSeconds	Interval between periodic reconciliations.
integer	
zookeeperSessionTimeoutSeconds	**The `zookeeperSessionTimeoutSeconds` property has been deprecated.** This property has been deprecated because ZooKeeper is not used anymore by the User Operator. Timeout for the ZooKeeper session.
integer	
secretPrefix	The prefix that will be added to the KafkaUser name to be used as the Secret name.
string	
livenessProbe	Pod liveness checking.
`Probe`	
readinessProbe	Pod readiness checking.
`Probe`	
resources	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
logging	Logging configuration. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
`InlineLogging`, `ExternalLogging`	

Property	Description
jvmOptions	JVM Options for pods.
`JvmOptions`	

4.2.47. `TlsSidecar` schema reference

Used in: `CruiseControlSpec`, `EntityOperatorSpec`

Full list of `TlsSidecar` schema properties

Configures a TLS sidecar, which is a container that runs in a pod, but serves a supporting purpose. In Strimzi, the TLS sidecar uses TLS to encrypt and decrypt communication between components and ZooKeeper.

The TLS sidecar is used in the Entity Operator.

The TLS sidecar is configured using the `tlsSidecar` property in `Kafka.spec.entityOperator`.

The TLS sidecar supports the following additional options:

- `image`
- `resources`
- `logLevel`
- `readinessProbe`
- `livenessProbe`

The `resources` property specifies the memory and CPU resources allocated for the TLS sidecar.

The `image` property configures the container image which will be used.

The `readinessProbe` and `livenessProbe` properties configure healthcheck probes for the TLS sidecar.

The `logLevel` property specifies the logging level. The following logging levels are supported:

- emerg
- alert
- crit
- err
- warning
- notice
- info
- debug

The default value is *notice*.

Example TLS sidecar configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  entityOperator:
    # ...
    tlsSidecar:
      resources:
        requests:
          cpu: 200m
          memory: 64Mi
        limits:
          cpu: 500m
          memory: 128Mi
    # ...
```

`TlsSidecar` schema properties

Property	Description
image	The docker image for the container.
string	
livenessProbe	Pod liveness checking.
`Probe`	
logLevel	The log level for the TLS sidecar. Default value is `notice`.
string (one of [emerg, debug, crit, err, alert, warning, notice, info])	
readinessProbe	Pod readiness checking.
`Probe`	
resources	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	

4.2.48. `EntityOperatorTemplate` schema reference

Used in: `EntityOperatorSpec`

Property	Description
deployment	Template for Entity Operator `Deployment`.
`DeploymentTemplate`	

Property	Description
pod PodTemplate	Template for Entity Operator Pods.
topicOperatorContainer ContainerTemplate	Template for the Entity Topic Operator container.
userOperatorContainer ContainerTemplate	Template for the Entity User Operator container.
tlsSidecarContainer ContainerTemplate	Template for the Entity Operator TLS sidecar container.
serviceAccount ResourceTemplate	Template for the Entity Operator service account.
entityOperatorRole ResourceTemplate	Template for the Entity Operator Role.
topicOperatorRoleBinding ResourceTemplate	Template for the Entity Topic Operator RoleBinding.
userOperatorRoleBinding ResourceTemplate	Template for the Entity Topic Operator RoleBinding.

4.2.49. DeploymentTemplate schema reference

Used in: CruiseControlTemplate, EntityOperatorTemplate, JmxTransTemplate, KafkaBridgeTemplate, KafkaConnectTemplate, KafkaExporterTemplate, KafkaMirrorMakerTemplate

Full list of DeploymentTemplate schema properties

Use deploymentStrategy to specify the strategy used to replace old pods with new ones when deployment configuration changes.

Use one of the following values:

- RollingUpdate: Pods are restarted with zero downtime.
- Recreate: Pods are terminated before new ones are created.

Using the Recreate deployment strategy has the advantage of not requiring spare resources, but the disadvantage is the application downtime.

Example showing the deployment strategy set to Recreate.

```
# ...
template:
  deployment:
    deploymentStrategy: Recreate
# ...
```

This configuration change does not cause a rolling update.

`DeploymentTemplate` schema properties

Property	Description
metadata `MetadataTemplate`	Metadata applied to the resource.
deploymentStrategy string (one of [RollingUpdate, Recreate])	Pod replacement strategy for deployment configuration changes. Valid values are `RollingUpdate` and `Recreate`. Defaults to `RollingUpdate`.

4.2.50. `CertificateAuthority` schema reference

Used in: `KafkaSpec`

Configuration of how TLS certificates are used within the cluster. This applies to certificates used for both internal communication within the cluster and to certificates used for client access via `Kafka.spec.kafka.listeners.tls`.

Property	Description
generateCertificateAuthority boolean	If true then Certificate Authority certificates will be generated automatically. Otherwise the user will need to provide a Secret with the CA certificate. Default is true.
generateSecretOwnerReference boolean	If `true`, the Cluster and Client CA Secrets are configured with the `ownerReference` set to the `Kafka` resource. If the `Kafka` resource is deleted when `true`, the CA Secrets are also deleted. If `false`, the `ownerReference` is disabled. If the `Kafka` resource is deleted when `false`, the CA Secrets are retained and available for reuse. Default is `true`.
validityDays integer	The number of days generated certificates should be valid for. The default is 365.
renewalDays integer	The number of days in the certificate renewal period. This is the number of days before the a certificate expires during which renewal actions may be performed. When `generateCertificateAuthority` is true, this will cause the generation of a new certificate. When `generateCertificateAuthority` is true, this will cause extra logging at WARN level about the pending certificate expiry. Default is 30.

Property	Description
certificateExpirationPolicy	How should CA certificate expiration be handled when `generateCertificateAuthority=true`. The default is for a new CA certificate to be generated reusing the existing private key.
string (one of [replace-key, renew-certificate])	

4.2.51. `CruiseControlSpec` schema reference

Used in: `KafkaSpec`

Full list of `CruiseControlSpec` schema properties

Configures a Cruise Control cluster.

Configuration options relate to:

- Goals configuration
- Capacity limits for resource distribution goals

`config`

Use the `config` properties to configure Cruise Control options as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the Cruise Control documentation.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Security (encryption, authentication, and authorization)
- Connection to the Kafka cluster
- Client ID configuration
- ZooKeeper connectivity
- Web server configuration
- Self healing

Properties with the following prefixes cannot be set:

- `bootstrap.servers`
- `capacity.config.file`

- client.id
- failed.brokers.zk.path
- kafka.broker.failure.detection.enable
- metric.reporter.sampler.bootstrap.servers
- network.
- request.reason.required
- security.
- self.healing.
- ssl.
- topic.config.provider.class
- two.step.
- webserver.accesslog.
- webserver.api.urlprefix
- webserver.http.
- webserver.session.path
- zookeeper.

If the config property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to Cruise Control, including the following exceptions to the options configured by Strimzi:

- Any ssl configuration for supported TLS versions and cipher suites
- Configuration for webserver properties to enable Cross-Origin Resource Sharing (CORS)

Example Cruise Control configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    config:
      # Note that `default.goals` (superset) must also include all `hard.goals`
(subset)
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal
      hard.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal
      cpu.balance.threshold: 1.1
      metadata.max.age.ms: 300000
```

```
      send.buffer.bytes: 131072
      webserver.http.cors.enabled: true
      webserver.http.cors.origin: "*"
      webserver.http.cors.exposeheaders: "User-Task-ID,Content-Type"
    # ...
```

Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) is a HTTP mechanism for controlling access to REST APIs. Restrictions can be on access methods or originating URLs of client applications. You can enable CORS with Cruise Control using the `webserver.http.cors.enabled` property in the `config`. When enabled, CORS permits read access to the Cruise Control REST API from applications that have different originating URLs than Strimzi. This allows applications from specified origins to use `GET` requests to fetch information about the Kafka cluster through the Cruise Control API. For example, applications can fetch information on the current cluster load or the most recent optimization proposal. `POST` requests are not permitted.

NOTE For more information on using CORS with Cruise Control, see REST APIs in the Cruise Control Wiki.

Enabling CORS for Cruise Control

You enable and configure CORS in `Kafka.spec.cruiseControl.config`.

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    config:
      webserver.http.cors.enabled: true ①
      webserver.http.cors.origin: "*" ②
      webserver.http.cors.exposeheaders: "User-Task-ID,Content-Type" ③

    # ...
```

① Enables CORS.

② Specifies permitted origins for the `Access-Control-Allow-Origin` HTTP response header. You can use a wildcard or specify a single origin as a URL. If you use a wildcard, a response is returned following requests from any origin.

③ Exposes specified header names for the `Access-Control-Expose-Headers` HTTP response header. Applications in permitted origins can read responses with the specified headers.

Cruise Control REST API security

The Cruise Control REST API is secured with HTTP Basic authentication and SSL to protect the cluster against potentially destructive Cruise Control operations, such as decommissioning Kafka brokers. We recommend that Cruise Control in Strimzi is **only used with these settings enabled**.

However, it is possible to disable these settings by specifying the following Cruise Control configuration:

- To disable the built-in HTTP Basic authentication, set `webserver.security.enable` to `false`.
- To disable the built-in SSL, set `webserver.ssl.enable` to `false`.

Cruise Control configuration to disable API authorization, authentication, and SSL

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    config:
      webserver.security.enable: false
      webserver.ssl.enable: false
# ...
```

brokerCapacity

Cruise Control uses capacity limits to determine if optimization goals for resource capacity limits are being broken. There are four goals of this type:

- `DiskCapacityGoal` - Disk utilization capacity
- `CpuCapacityGoal` - CPU utilization capacity
- `NetworkInboundCapacityGoal` - Network inbound utilization capacity
- `NetworkOutboundCapacityGoal` - Network outbound utilization capacity

You specify capacity limits for Kafka broker resources in the `brokerCapacity` property in `Kafka.spec.cruiseControl` . They are enabled by default and you can change their default values. Capacity limits can be set for the following broker resources:

- `cpu` - CPU resource in millicores or CPU cores (Default: 1)
- `inboundNetwork` - Inbound network throughput in byte units per second (Default: 10000KiB/s)
- `outboundNetwork` - Outbound network throughput in byte units per second (Default: 10000KiB/s)

For network throughput, use an integer value with standard Kubernetes byte units (K, M, G) or their bibyte (power of two) equivalents (Ki, Mi, Gi) per second.

NOTE	Disk and CPU capacity limits are automatically generated by Strimzi, so you do not

> need to set them. In order to guarantee accurate rebalance proposals when using CPU goals, you can set CPU requests equal to CPU limits in `Kafka.spec.kafka.resources`. That way, all CPU resources are reserved upfront and are always available. This configuration allows Cruise Control to properly evaluate the CPU utilization when preparing the rebalance proposals based on CPU goals. In cases where you cannot set CPU requests equal to CPU limits in `Kafka.spec.kafka.resources`, you can set the CPU capacity manually for the same accuracy.

Example Cruise Control brokerCapacity configuration using bibyte units

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      cpu: "2"
      inboundNetwork: 10000KiB/s
      outboundNetwork: 10000KiB/s
    # ...
```

Capacity overrides

Brokers might be running on nodes with heterogeneous network or CPU resources. If that's the case, specify `overrides` that set the network capacity and CPU limits for each broker. The overrides ensure an accurate rebalance between the brokers. Override capacity limits can be set for the following broker resources:

- `cpu` - CPU resource in millicores or CPU cores (Default: 1)
- `inboundNetwork` - Inbound network throughput in byte units per second (Default: 10000KiB/s)
- `outboundNetwork` - Outbound network throughput in byte units per second (Default: 10000KiB/s)

An example of Cruise Control capacity overrides configuration using bibyte units

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      cpu: "1"
      inboundNetwork: 10000KiB/s
```

```
      outboundNetwork: 10000KiB/s
      overrides:
      - brokers: [0]
        cpu: "2.755"
        inboundNetwork: 20000KiB/s
        outboundNetwork: 20000KiB/s
      - brokers: [1, 2]
        cpu: 3000m
        inboundNetwork: 30000KiB/s
        outboundNetwork: 30000KiB/s
```

For more information, refer to the BrokerCapacity schema reference.

Logging configuration

Cruise Control has its own configurable logger:

- `rootLogger.level`

Cruise Control uses the Apache `log4j2` logger implementation.

Use the `logging` property to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. Inside the ConfigMap, the logging configuration is described using `log4j.properties`. Both `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. Here we see examples of `inline` and `external` logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
  cruiseControl:
    # ...
    logging:
      type: inline
      loggers:
        rootLogger.level: "INFO"
    # ...
```

External logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
  cruiseControl:
    # ...
    logging:
      type: external
      valueFrom:
        configMapKeyRef:
          name: customConfigMap
          key: cruise-control-log4j.properties
    # ...
```

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`CruiseControlSpec` schema properties

Property	Description
image	The docker image for the pods.
string	
tlsSidecar	**The `tlsSidecar` property has been deprecated.** TLS sidecar configuration.
`TlsSidecar`	
resources	CPU and memory resources to reserve for the Cruise Control container. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
livenessProbe	Pod liveness checking for the Cruise Control container.
`Probe`	
readinessProbe	Pod readiness checking for the Cruise Control container.
`Probe`	
jvmOptions	JVM Options for the Cruise Control container.
`JvmOptions`	
logging	Logging configuration (Log4j 2) for Cruise Control. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
`InlineLogging`, `ExternalLogging`	
template	Template to specify how Cruise Control resources, `Deployments` and `Pods`, are generated.
`CruiseControlTemplate`	

Property	Description
brokerCapacity `BrokerCapacity`	The Cruise Control `brokerCapacity` configuration.
config	The Cruise Control configuration. For a full list of configuration options refer to https://github.com/linkedin/cruise-control/wiki/Configurations. Note that properties with the following prefixes cannot be set: bootstrap.servers, client.id, zookeeper., network., security., failed.brokers.zk.path,webserver.http., webserver.api.urlprefix, webserver.session.path, webserver.accesslog., two.step., request.reason.required,metric.reporter.sampler.bootstrap.servers, capacity.config.file, self.healing., ssl., kafka.broker.failure.detection.enable, topic.config.provider.class (with the exception of: ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols, webserver.http.cors.enabled, webserver.http.cors.origin, webserver.http.cors.exposeheaders, webserver.security.enable, webserver.ssl.enable).
map	
metricsConfig `JmxPrometheusExporterMetrics`	Metrics configuration. The type depends on the value of the `metricsConfig.type` property within the given object, which must be one of [jmxPrometheusExporter].

4.2.52. `CruiseControlTemplate` schema reference

Used in: `CruiseControlSpec`

Property	Description
deployment `DeploymentTemplate`	Template for Cruise Control `Deployment`.
pod `PodTemplate`	Template for Cruise Control `Pods`.
apiService `InternalServiceTemplate`	Template for Cruise Control API `Service`.

Property	Description
podDisruptionBudget `PodDisruptionBudgetTemplate`	Template for Cruise Control `PodDisruptionBudget`.
cruiseControlContainer `ContainerTemplate`	Template for the Cruise Control container.
tlsSidecarContainer `ContainerTemplate`	**The `tlsSidecarContainer` property has been deprecated.** Template for the Cruise Control TLS sidecar container.
serviceAccount `ResourceTemplate`	Template for the Cruise Control service account.

4.2.53. `BrokerCapacity` schema reference

Used in: `CruiseControlSpec`

Property	Description
disk string	**The `disk` property has been deprecated.** The Cruise Control disk capacity setting has been deprecated, is ignored, and will be removed in the future Broker capacity for disk in bytes. Use a number value with either standard Kubernetes byte units (K, M, G, or T), their bibyte (power of two) equivalents (Ki, Mi, Gi, or Ti), or a byte value with or without E notation. For example, 100000M, 100000Mi, 104857600000, or 1e+11.
cpuUtilization integer	**The `cpuUtilization` property has been deprecated.** The Cruise Control CPU capacity setting has been deprecated, is ignored, and will be removed in the future Broker capacity for CPU resource utilization as a percentage (0 - 100).
cpu string	Broker capacity for CPU resource in cores or millicores. For example, 1, 1.500, 1500m. For more information on valid CPU resource units see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu.
inboundNetwork string	Broker capacity for inbound network throughput in bytes per second. Use an integer value with standard Kubernetes byte units (K, M, G) or their bibyte (power of two) equivalents (Ki, Mi, Gi) per second. For example, 10000KiB/s.

Property	Description
outboundNetwork string	Broker capacity for outbound network throughput in bytes per second. Use an integer value with standard Kubernetes byte units (K, M, G) or their bibyte (power of two) equivalents (Ki, Mi, Gi) per second. For example, 10000KiB/s.
overrides `BrokerCapacityOverride` array	Overrides for individual brokers. The `overrides` property lets you specify a different capacity configuration for different brokers.

4.2.54. `BrokerCapacityOverride` schema reference

Used in: `BrokerCapacity`

Property	Description
brokers integer array	List of Kafka brokers (broker identifiers).
cpu string	Broker capacity for CPU resource in cores or millicores. For example, 1, 1.500, 1500m. For more information on valid CPU resource units see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu.
inboundNetwork string	Broker capacity for inbound network throughput in bytes per second. Use an integer value with standard Kubernetes byte units (K, M, G) or their bibyte (power of two) equivalents (Ki, Mi, Gi) per second. For example, 10000KiB/s.
outboundNetwork string	Broker capacity for outbound network throughput in bytes per second. Use an integer value with standard Kubernetes byte units (K, M, G) or their bibyte (power of two) equivalents (Ki, Mi, Gi) per second. For example, 10000KiB/s.

4.2.55. `JmxTransSpec` schema reference

The type `JmxTransSpec` has been deprecated.

Used in: `KafkaSpec`

Property	Description
image string	The image to use for the JmxTrans.

Property	Description
outputDefinitions `JmxTransOutputDefinitionTemplate` array	Defines the output hosts that will be referenced later on. For more information on these properties see, `JmxTransOutputDefinitionTemplate` schema reference.
logLevel string	Sets the logging level of the JmxTrans deployment.For more information see, JmxTrans Logging Level.
kafkaQueries `JmxTransQueryTemplate` array	Queries to send to the Kafka brokers to define what data should be read from each broker. For more information on these properties see, `JmxTransQueryTemplate` schema reference.
resources ResourceRequirements	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
template `JmxTransTemplate`	Template for JmxTrans resources.

4.2.56. `JmxTransOutputDefinitionTemplate` schema reference

Used in: `JmxTransSpec`

Property	Description
outputType string	Template for setting the format of the data that will be pushed.For more information see JmxTrans OutputWriters.
host string	The DNS/hostname of the remote host that the data is pushed to.
port integer	The port of the remote host that the data is pushed to.
flushDelayInSeconds integer	How many seconds the JmxTrans waits before pushing a new set of data out.
typeNames string array	Template for filtering data to be included in response to a wildcard query. For more information see JmxTrans queries.
name string	Template for setting the name of the output definition. This is used to identify where to send the results of queries should be sent.

4.2.57. JmxTransQueryTemplate schema reference

Used in: JmxTransSpec

Property	Description
targetMBean string	If using wildcards instead of a specific MBean then the data is gathered from multiple MBeans. Otherwise if specifying an MBean then data is gathered from that specified MBean.
attributes string array	Determine which attributes of the targeted MBean should be included.
outputs string array	List of the names of output definitions specified in the spec.kafka.jmxTrans.outputDefinitions that have defined where JMX metrics are pushed to, and in which data format.

4.2.58. JmxTransTemplate schema reference

Used in: JmxTransSpec

Property	Description
deployment DeploymentTemplate	Template for JmxTrans Deployment.
pod PodTemplate	Template for JmxTrans Pods.
container ContainerTemplate	Template for JmxTrans container.
serviceAccount ResourceTemplate	Template for the JmxTrans service account.

4.2.59. KafkaExporterSpec schema reference

Used in: KafkaSpec

Property	Description
image string	The docker image for the pods.
groupRegex string	Regular expression to specify which consumer groups to collect. Default value is .*.
topicRegex string	Regular expression to specify which topics to collect. Default value is .*.

Property	Description
resources ResourceRequirements	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
logging string	Only log messages with the given severity or above. Valid levels: [`info`, `debug`, `trace`]. Default log level is `info`.
enableSaramaLogging boolean	Enable Sarama logging, a Go client library used by the Kafka Exporter.
template `KafkaExporterTemplate`	Customization of deployment templates and pods.
livenessProbe `Probe`	Pod liveness check.
readinessProbe `Probe`	Pod readiness check.

4.2.60. `KafkaExporterTemplate` schema reference

Used in: `KafkaExporterSpec`

Property	Description
deployment `DeploymentTemplate`	Template for Kafka Exporter `Deployment`.
pod `PodTemplate`	Template for Kafka Exporter `Pods`.
service `ResourceTemplate`	**The `service` property has been deprecated.** The Kafka Exporter service has been removed. Template for Kafka Exporter `Service`.
container `ContainerTemplate`	Template for the Kafka Exporter container.
serviceAccount `ResourceTemplate`	Template for the Kafka Exporter service account.

4.2.61. `KafkaStatus` schema reference

Used in: `Kafka`

Property	Description
conditions `Condition` array	List of status conditions.

Property	Description
observedGeneration integer	The generation of the CRD that was last reconciled by the operator.
listeners `ListenerStatus` array	Addresses of the internal and external listeners.
clusterId string	Kafka cluster Id.

4.2.62. `Condition` schema reference

Used in: `KafkaBridgeStatus`, `KafkaConnectorStatus`, `KafkaConnectStatus`, `KafkaMirrorMaker2Status`, `KafkaMirrorMakerStatus`, `KafkaRebalanceStatus`, `KafkaStatus`, `KafkaTopicStatus`, `KafkaUserStatus`

Property	Description
type string	The unique identifier of a condition, used to distinguish between other conditions in the resource.
status string	The status of the condition, either True, False or Unknown.
lastTransitionTime string	Last time the condition of a type changed from one status to another. The required format is 'yyyy-MM-ddTHH:mm:ssZ', in the UTC time zone.
reason string	The reason for the condition's last transition (a single word in CamelCase).
message string	Human-readable message indicating details about the condition's last transition.

4.2.63. `ListenerStatus` schema reference

Used in: `KafkaStatus`

Property	Description
type string	**The `type` property has been deprecated, and should now be configured using `name`.** The name of the listener.
name string	The name of the listener.
addresses `ListenerAddress` array	A list of the addresses for this listener.

Property	Description
bootstrapServers string	A comma-separated list of `host:port` pairs for connecting to the Kafka cluster using this listener.
certificates string array	A list of TLS certificates which can be used to verify the identity of the server when connecting to the given listener. Set only for `tls` and `external` listeners.

4.2.64. `ListenerAddress` schema reference

Used in: `ListenerStatus`

Property	Description
host string	The DNS name or IP address of the Kafka bootstrap service.
port integer	The port of the Kafka bootstrap service.

4.2.65. `KafkaConnect` schema reference

Property	Description
spec `KafkaConnectSpec`	The specification of the Kafka Connect cluster.
status `KafkaConnectStatus`	The status of the Kafka Connect cluster.

4.2.66. `KafkaConnectSpec` schema reference

Used in: `KafkaConnect`

Full list of `KafkaConnectSpec` schema properties

Configures a Kafka Connect cluster.

`config`

Use the `config` properties to configure Kafka Connect options as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Certain options have default values:

- `group.id` with default value `connect-cluster`
- `offset.storage.topic` with default value `connect-cluster-offsets`
- `config.storage.topic` with default value `connect-cluster-configs`
- `status.storage.topic` with default value `connect-cluster-status`
- `key.converter` with default value `org.apache.kafka.connect.json.JsonConverter`
- `value.converter` with default value `org.apache.kafka.connect.json.JsonConverter`

These options are automatically configured in case they are not present in the `KafkaConnect.spec.config` properties.

Exceptions

You can specify and configure the options listed in the Apache Kafka documentation.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Kafka cluster bootstrap address
- Security (encryption, authentication, and authorization)
- Listener and REST interface configuration
- Plugin path configuration

Properties with the following prefixes cannot be set:

- `bootstrap.servers`
- `consumer.interceptor.classes`
- `listeners.`
- `plugin.path`
- `producer.interceptor.classes`
- `rest.`
- `sasl.`
- `security.`
- `ssl.`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to Kafka Connect, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites

Example Kafka Connect configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    config.storage.replication.factor: 3
    offset.storage.replication.factor: 3
    status.storage.replication.factor: 3
  # ...
```

IMPORTANT	The Cluster Operator does not validate keys or values in the `config` object provided. If an invalid configuration is provided, the Kafka Connect cluster might not start or might become unstable. In this case, fix the configuration so that the Cluster Operator can roll out the new configuration to all Kafka Connect nodes.

`logging`

Kafka Connect has its own configurable loggers:

- `connect.root.logger.level`
- `log4j.logger.org.reflections`

Further loggers are added depending on the Kafka Connect plugins running.

Use a curl request to get a complete list of Kafka Connect loggers running from any Kafka broker pod:

```
curl -s http://<connect-cluster-name>-connect-api:8083/admin/loggers/
```

Kafka Connect uses the Apache `log4j` logger implementation.

Use the `logging` property to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. Inside the

ConfigMap, the logging configuration is described using `log4j.properties`. Both `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. For more information about log levels, see Apache logging services.

Here we see examples of `inline` and `external` logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
spec:
  # ...
  logging:
    type: inline
    loggers:
      connect.root.logger.level: "INFO"
  # ...
```

External logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: customConfigMap
        key: connect-logging.log4j
  # ...
```

Any available loggers that are not configured have their level set to `OFF`.

If Kafka Connect was deployed using the Cluster Operator, changes to Kafka Connect logging levels are applied dynamically.

If you use external logging, a rolling update is triggered when logging appenders are changed.

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`KafkaConnectSpec` schema properties

Property	Description
version	The Kafka Connect version. Defaults to 3.4.0. Consult the user documentation to understand the process required to upgrade or downgrade the version.
string	
replicas	The number of pods in the Kafka Connect group. Defaults to 3.
integer	
image	The docker image for the pods.
string	
bootstrapServers	Bootstrap servers to connect to. This should be given as a comma separated list of <*hostname*>:_<port>_ pairs.
string	
tls	TLS configuration.
`ClientTls`	
authentication	Authentication configuration for Kafka Connect. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, scram-sha-256, scram-sha-512, plain, oauth].
`KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationOAuth`	
config	The Kafka Connect configuration. Properties with the following prefixes cannot be set: ssl., sasl., security., listeners, plugin.path, rest., bootstrap.servers, consumer.interceptor.classes, producer.interceptor.classes (with the exception of: ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols).
map	
resources	The maximum limits for CPU and memory resources and the requested initial resources. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
livenessProbe	Pod liveness checking.
`Probe`	
readinessProbe	Pod readiness checking.
`Probe`	
jvmOptions	JVM Options for pods.
`JvmOptions`	

Property	Description
jmxOptions KafkaJmxOptions	JMX Options.
logging InlineLogging, ExternalLogging	Logging configuration for Kafka Connect. The type depends on the value of the logging.type property within the given object, which must be one of [inline, external].
clientRackInitImage string	The image of the init container used for initializing the client.rack.
rack Rack	Configuration of the node label which will be used as the client.rack consumer configuration.
tracing JaegerTracing, OpenTelemetryTracing	The configuration of tracing in Kafka Connect. The type depends on the value of the tracing.type property within the given object, which must be one of [jaeger, opentelemetry].
template KafkaConnectTemplate	Template for Kafka Connect and Kafka Mirror Maker 2 resources. The template allows users to specify how the Deployment, Pods and Service are generated.
externalConfiguration ExternalConfiguration	Pass data from Secrets or ConfigMaps to the Kafka Connect pods and use them to configure connectors.
build Build	Configures how the Connect container image should be built. Optional.
metricsConfig JmxPrometheusExporterMetrics	Metrics configuration. The type depends on the value of the metricsConfig.type property within the given object, which must be one of [jmxPrometheusExporter].

4.2.67. ClientTls schema reference

Used in: KafkaBridgeSpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

Full list of ClientTls schema properties

Configures TLS trusted certificates for connecting KafkaConnect, KafkaBridge, KafkaMirror, KafkaMirrorMaker2 to the cluster.

trustedCertificates

Provide a list of secrets using the trustedCertificates property.

`ClientTls` schema properties

Property	Description
trustedCertificates	Trusted certificates for TLS connection.
`CertSecretSource` array	

4.2.68. `KafkaClientAuthenticationTls` schema reference

Used in: `KafkaBridgeSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2ClusterSpec`, `KafkaMirrorMakerConsumerSpec`, `KafkaMirrorMakerProducerSpec`

Full list of `KafkaClientAuthenticationTls` schema properties

To configure mTLS authentication, set the `type` property to the value `tls`. mTLS uses a TLS certificate to authenticate.

`certificateAndKey`

The certificate is specified in the `certificateAndKey` property and is always loaded from a Kubernetes secret. In the secret, the certificate must be stored in X509 format under two different keys: public and private.

You can use the secrets created by the User Operator, or you can create your own TLS certificate file, with the keys used for authentication, then create a `Secret` from the file:

```
kubectl create secret generic MY-SECRET \
--from-file=MY-PUBLIC-TLS-CERTIFICATE-FILE.crt \
--from-file=MY-PRIVATE.key
```

NOTE mTLS authentication can only be used with TLS connections.

Example mTLS configuration

```
authentication:
  type: tls
  certificateAndKey:
    secretName: my-secret
    certificate: my-public-tls-certificate-file.crt
    key: private.key
```

`KafkaClientAuthenticationTls` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaClientAuthenticationTls` type from `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationOAuth`. It must have the value `tls` for the type `KafkaClientAuthenticationTls`.

Property	Description
certificateAndKey CertAndKeySecretSource	Reference to the Secret which holds the certificate and private key pair.
type string	Must be tls.

4.2.69. KafkaClientAuthenticationScramSha256 schema reference

Used in: KafkaBridgeSpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

Full list of KafkaClientAuthenticationScramSha256 schema properties

To configure SASL-based SCRAM-SHA-256 authentication, set the type property to scram-sha-256. The SCRAM-SHA-256 authentication mechanism requires a username and password.

username

Specify the username in the username property.

passwordSecret

In the passwordSecret property, specify a link to a Secret containing the password.

You can use the secrets created by the User Operator.

If required, you can create a text file that contains the password, in cleartext, to use for authentication:

```
echo -n PASSWORD > MY-PASSWORD.txt
```

You can then create a Secret from the text file, setting your own field name (key) for the password:

```
kubectl create secret generic MY-CONNECT-SECRET-NAME --from-file=MY-PASSWORD-FIELD-NAME=./MY-PASSWORD.txt
```

Example Secret for SCRAM-SHA-256 client authentication for Kafka Connect

```
apiVersion: v1
kind: Secret
metadata:
  name: my-connect-secret-name
type: Opaque
data:
  my-connect-password-field: LFTIyFRFlMmU2N2Tm
```

The secretName property contains the name of the Secret, and the password property contains the

name of the key under which the password is stored inside the `Secret`.

IMPORTANT	Do not specify the actual password in the `password` property.

Example SASL-based SCRAM-SHA-256 client authentication configuration for Kafka Connect

```
authentication:
  type: scram-sha-256
  username: my-connect-username
  passwordSecret:
    secretName: my-connect-secret-name
    password: my-connect-password-field
```

`KafkaClientAuthenticationScramSha256` schema properties

Property	Description
passwordSecret `PasswordSecretSource`	Reference to the `Secret` which holds the password.
type string	Must be `scram-sha-256`.
username string	Username used for the authentication.

4.2.70. `PasswordSecretSource` schema reference

Used in: `KafkaClientAuthenticationOAuth`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`

Property	Description
password string	The name of the key in the Secret under which the password is stored.
secretName string	The name of the Secret containing the password.

4.2.71. `KafkaClientAuthenticationScramSha512` schema reference

Used in: `KafkaBridgeSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2ClusterSpec`, `KafkaMirrorMakerConsumerSpec`, `KafkaMirrorMakerProducerSpec`

Full list of `KafkaClientAuthenticationScramSha512` schema properties

To configure SASL-based SCRAM-SHA-512 authentication, set the `type` property to `scram-sha-512`. The SCRAM-SHA-512 authentication mechanism requires a username and password.

`username`

Specify the username in the `username` property.

`passwordSecret`

In the `passwordSecret` property, specify a link to a `Secret` containing the password.

You can use the secrets created by the User Operator.

If required, you can create a text file that contains the password, in cleartext, to use for authentication:

```
echo -n PASSWORD > MY-PASSWORD.txt
```

You can then create a `Secret` from the text file, setting your own field name (key) for the password:

```
kubectl create secret generic MY-CONNECT-SECRET-NAME --from-file=MY-PASSWORD-FIELD-
NAME=./MY-PASSWORD.txt
```

Example Secret for SCRAM-SHA-512 client authentication for Kafka Connect

```
apiVersion: v1
kind: Secret
metadata:
  name: my-connect-secret-name
type: Opaque
data:
  my-connect-password-field: LFTIyFRFlMmU2N2Tm
```

The `secretName` property contains the name of the `Secret`, and the `password` property contains the name of the key under which the password is stored inside the `Secret`.

IMPORTANT	Do not specify the actual password in the `password` property.

Example SASL-based SCRAM-SHA-512 client authentication configuration for Kafka Connect

```
authentication:
  type: scram-sha-512
  username: my-connect-username
  passwordSecret:
    secretName: my-connect-secret-name
    password: my-connect-password-field
```

`KafkaClientAuthenticationScramSha512` **schema properties**

Property	Description
passwordSecret PasswordSecretSource	Reference to the Secret which holds the password.
type string	Must be scram-sha-512.
username string	Username used for the authentication.

4.2.72. KafkaClientAuthenticationPlain schema reference

Used in: KafkaBridgeSpec, KafkaConnectSpec, KafkaMirrorMaker2ClusterSpec, KafkaMirrorMakerConsumerSpec, KafkaMirrorMakerProducerSpec

Full list of KafkaClientAuthenticationPlain schema properties

To configure SASL-based PLAIN authentication, set the type property to plain. SASL PLAIN authentication mechanism requires a username and password.

WARNING	The SASL PLAIN mechanism will transfer the username and password across the network in cleartext. Only use SASL PLAIN authentication if TLS encryption is enabled.

username

Specify the username in the username property.

passwordSecret

In the passwordSecret property, specify a link to a Secret containing the password.

You can use the secrets created by the User Operator.

If required, create a text file that contains the password, in cleartext, to use for authentication:

```
echo -n PASSWORD > MY-PASSWORD.txt
```

You can then create a Secret from the text file, setting your own field name (key) for the password:

```
kubectl create secret generic MY-CONNECT-SECRET-NAME --from-file=MY-PASSWORD-FIELD-NAME=./MY-PASSWORD.txt
```

Example Secret for PLAIN client authentication for Kafka Connect

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: my-connect-secret-name
type: Opaque
data:
  my-password-field-name: LFTIyFRFlMmU2N2Tm
```

The `secretName` property contains the name of the `Secret` and the `password` property contains the name of the key under which the password is stored inside the `Secret`.

IMPORTANT	Do not specify the actual password in the `password` property.

An example SASL based PLAIN client authentication configuration

```
authentication:
  type: plain
  username: my-connect-username
  passwordSecret:
    secretName: my-connect-secret-name
    password: my-password-field-name
```

`KafkaClientAuthenticationPlain` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaClientAuthenticationPlain` type from `KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationOAuth`. It must have the value `plain` for the type `KafkaClientAuthenticationPlain`.

Property	Description
passwordSecret `PasswordSecretSource`	Reference to the `Secret` which holds the password.
type string	Must be `plain`.
username string	Username used for the authentication.

4.2.73. `KafkaClientAuthenticationOAuth` schema reference

Used in: `KafkaBridgeSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2ClusterSpec`, `KafkaMirrorMakerConsumerSpec`, `KafkaMirrorMakerProducerSpec`

Full list of `KafkaClientAuthenticationOAuth` schema properties

To configure OAuth client authentication, set the `type` property to `oauth`.

OAuth authentication can be configured using one of the following options:

- Client ID and secret

- Client ID and refresh token
- Access token
- Username and password
- TLS

Client ID and secret

You can configure the address of your authorization server in the `tokenEndpointUri` property together with the client ID and client secret used in authentication. The OAuth client will connect to the OAuth server, authenticate using the client ID and secret and get an access token which it will use to authenticate with the Kafka broker. In the `clientSecret` property, specify a link to a `Secret` containing the client secret.

An example of OAuth client authentication using client ID and client secret

```
authentication:
  type: oauth
  tokenEndpointUri:
https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  clientSecret:
    secretName: my-client-oauth-secret
    key: client-secret
```

Optionally, `scope` and `audience` can be specified if needed.

Client ID and refresh token

You can configure the address of your OAuth server in the `tokenEndpointUri` property together with the OAuth client ID and refresh token. The OAuth client will connect to the OAuth server, authenticate using the client ID and refresh token and get an access token which it will use to authenticate with the Kafka broker. In the `refreshToken` property, specify a link to a `Secret` containing the refresh token.

An example of OAuth client authentication using client ID and refresh token

```
authentication:
  type: oauth
  tokenEndpointUri:
https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  refreshToken:
    secretName: my-refresh-token-secret
    key: refresh-token
```

Access token

You can configure the access token used for authentication with the Kafka broker directly. In this case, you do not specify the `tokenEndpointUri`. In the `accessToken` property, specify a link to a `Secret` containing the access token.

An example of OAuth client authentication using only an access token

```
authentication:
  type: oauth
  accessToken:
    secretName: my-access-token-secret
    key: access-token
```

Username and password

OAuth username and password configuration uses the OAuth *Resource Owner Password Grant* mechanism. The mechanism is deprecated, and is only supported to enable integration in environments where client credentials (ID and secret) cannot be used. You might need to use user accounts if your access management system does not support another approach or user accounts are required for authentication.

A typical approach is to create a special user account in your authorization server that represents your client application. You then give the account a long randomly generated password and a very limited set of permissions. For example, the account can only connect to your Kafka cluster, but is not allowed to use any other services or login to the user interface.

Consider using a refresh token mechanism first.

You can configure the address of your authorization server in the `tokenEndpointUri` property together with the client ID, username and the password used in authentication. The OAuth client will connect to the OAuth server, authenticate using the username, the password, the client ID, and optionally even the client secret to obtain an access token which it will use to authenticate with the Kafka broker.

In the `passwordSecret` property, specify a link to a `Secret` containing the password.

Normally, you also have to configure a `clientId` using a public OAuth client. If you are using a confidential OAuth client, you also have to configure a `clientSecret`.

An example of OAuth client authentication using username and a password with a public client

```
authentication:
  type: oauth
  tokenEndpointUri:
https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  username: my-username
  passwordSecret:
    secretName: my-password-secret-name
    password: my-password-field-name
  clientId: my-public-client-id
```

An example of OAuth client authentication using a username and a password with a confidential client

```
authentication:
  type: oauth
```

```
  tokenEndpointUri:
https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  username: my-username
  passwordSecret:
    secretName: my-password-secret-name
    password: my-password-field-name
  clientId: my-confidential-client-id
  clientSecret:
    secretName: my-confidential-client-oauth-secret
    key: client-secret
```

Optionally, `scope` and `audience` can be specified if needed.

TLS

Accessing the OAuth server using the HTTPS protocol does not require any additional configuration as long as the TLS certificates used by it are signed by a trusted certification authority and its hostname is listed in the certificate.

If your OAuth server is using certificates which are self-signed or are signed by a certification authority which is not trusted, you can configure a list of trusted certificates in the custom resource. The `tlsTrustedCertificates` property contains a list of secrets with key names under which the certificates are stored. The certificates must be stored in X509 format.

An example of TLS certificates provided

```
authentication:
  type: oauth
  tokenEndpointUri:
https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  refreshToken:
    secretName: my-refresh-token-secret
    key: refresh-token
  tlsTrustedCertificates:
    - secretName: oauth-server-ca
      certificate: tls.crt
```

The OAuth client will by default verify that the hostname of your OAuth server matches either the certificate subject or one of the alternative DNS names. If it is not required, you can disable the hostname verification.

An example of disabled TLS hostname verification

```
authentication:
  type: oauth
  tokenEndpointUri:
https://sso.myproject.svc:8443/auth/realms/internal/protocol/openid-connect/token
  clientId: my-client-id
  refreshToken:
```

```
    secretName: my-refresh-token-secret
    key: refresh-token
  disableTlsHostnameVerification: true
```

`KafkaClientAuthenticationOAuth` schema properties

The `type` property is a discriminator that distinguishes use of the `KafkaClientAuthenticationOAuth` type from `KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`. It must have the value `oauth` for the type `KafkaClientAuthenticationOAuth`.

Property	Description
accessToken `GenericSecretSource`	Link to Kubernetes Secret containing the access token which was obtained from the authorization server.
accessTokenIsJwt boolean	Configure whether access token should be treated as JWT. This should be set to `false` if the authorization server returns opaque tokens. Defaults to `true`.
audience string	OAuth audience to use when authenticating against the authorization server. Some authorization servers require the audience to be explicitly set. The possible values depend on how the authorization server is configured. By default, `audience` is not specified when performing the token endpoint request.
clientId string	OAuth Client ID which the Kafka client can use to authenticate against the OAuth server and use the token endpoint URI.
clientSecret `GenericSecretSource`	Link to Kubernetes Secret containing the OAuth client secret which the Kafka client can use to authenticate against the OAuth server and use the token endpoint URI.
connectTimeoutSeconds integer	The connect timeout in seconds when connecting to authorization server. If not set, the effective connect timeout is 60 seconds.
disableTlsHostnameVerification boolean	Enable or disable TLS hostname verification. Default value is `false`.
enableMetrics boolean	Enable or disable OAuth metrics. Default value is `false`.
httpRetries integer	The maximum number of retries to attempt if an initial HTTP request fails. If not set, the default is to not attempt any retries.

Property	Description
httpRetryPauseMs integer	The pause to take before retrying a failed HTTP request. If not set, the default is to not pause at all but to immediately repeat a request.
maxTokenExpirySeconds integer	Set or limit time-to-live of the access tokens to the specified number of seconds. This should be set if the authorization server returns opaque tokens.
passwordSecret `PasswordSecretSource`	Reference to the `Secret` which holds the password.
readTimeoutSeconds integer	The read timeout in seconds when connecting to authorization server. If not set, the effective read timeout is 60 seconds.
refreshToken `GenericSecretSource`	Link to Kubernetes Secret containing the refresh token which can be used to obtain access token from the authorization server.
scope string	OAuth scope to use when authenticating against the authorization server. Some authorization servers require this to be set. The possible values depend on how authorization server is configured. By default `scope` is not specified when doing the token endpoint request.
tlsTrustedCertificates `CertSecretSource` array	Trusted certificates for TLS connection to the OAuth server.
tokenEndpointUri string	Authorization server token endpoint URI.
type string	Must be `oauth`.
username string	Username used for the authentication.

4.2.74. `JaegerTracing` schema reference

The type `JaegerTracing` has been deprecated.

Used in: `KafkaBridgeSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`, `KafkaMirrorMakerSpec`

The `type` property is a discriminator that distinguishes use of the `JaegerTracing` type from `OpenTelemetryTracing`. It must have the value `jaeger` for the type `JaegerTracing`.

Property	Description
type	Must be `jaeger`.
string	

4.2.75. `OpenTelemetryTracing` schema reference

Used in: `KafkaBridgeSpec`, `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`, `KafkaMirrorMakerSpec`

The `type` property is a discriminator that distinguishes use of the `OpenTelemetryTracing` type from `JaegerTracing`. It must have the value `opentelemetry` for the type `OpenTelemetryTracing`.

Property	Description
type	Must be `opentelemetry`.
string	

4.2.76. `KafkaConnectTemplate` schema reference

Used in: `KafkaConnectSpec`, `KafkaMirrorMaker2Spec`

Property	Description
deployment	Template for Kafka Connect `Deployment`.
`DeploymentTemplate`	
podSet	Template for Kafka Connect `StrimziPodSet` resource.
`ResourceTemplate`	
pod	Template for Kafka Connect `Pods`.
`PodTemplate`	
apiService	Template for Kafka Connect API `Service`.
`InternalServiceTemplate`	
headlessService	Template for Kafka Connect headless `Service`.
`InternalServiceTemplate`	
connectContainer	Template for the Kafka Connect container.
`ContainerTemplate`	
initContainer	Template for the Kafka init container.
`ContainerTemplate`	
podDisruptionBudget	Template for Kafka Connect `PodDisruptionBudget`.
`PodDisruptionBudgetTemplate`	
serviceAccount	Template for the Kafka Connect service account.
`ResourceTemplate`	
clusterRoleBinding	Template for the Kafka Connect ClusterRoleBinding.
`ResourceTemplate`	

Property	Description
buildPod PodTemplate	Template for Kafka Connect Build Pods. The build pod is used only on Kubernetes.
buildContainer ContainerTemplate	Template for the Kafka Connect Build container. The build container is used only on Kubernetes.
buildConfig BuildConfigTemplate	Template for the Kafka Connect BuildConfig used to build new container images. The BuildConfig is used only on OpenShift.
buildServiceAccount ResourceTemplate	Template for the Kafka Connect Build service account.
jmxSecret ResourceTemplate	Template for Secret of the Kafka Connect Cluster JMX authentication.

4.2.77. BuildConfigTemplate schema reference

Used in: KafkaConnectTemplate

Property	Description
metadata MetadataTemplate	Metadata to apply to the PodDisruptionBudgetTemplate resource.
pullSecret string	Container Registry Secret with the credentials for pulling the base image.

4.2.78. ExternalConfiguration schema reference

Used in: KafkaConnectSpec, KafkaMirrorMaker2Spec

Full list of ExternalConfiguration schema properties

Configures external storage properties that define configuration options for Kafka Connect connectors.

You can mount ConfigMaps or Secrets into a Kafka Connect pod as environment variables or volumes. Volumes and environment variables are configured in the externalConfiguration property in KafkaConnect.spec or KafkaMirrorMaker2.spec.

When applied, the environment variables and volumes are available for use when developing your connectors.

For more information, see Loading configuration values from external sources.

ExternalConfiguration schema properties

Property	Description
env `ExternalConfigurationEnv` array	Makes data from a Secret or ConfigMap available in the Kafka Connect pods as environment variables.
volumes `ExternalConfigurationVolumeSource` array	Makes data from a Secret or ConfigMap available in the Kafka Connect pods as volumes.

4.2.79. `ExternalConfigurationEnv` schema reference

Used in: `ExternalConfiguration`

Property	Description
name string	Name of the environment variable which will be passed to the Kafka Connect pods. The name of the environment variable cannot start with `KAFKA_` or `STRIMZI_`.
valueFrom `ExternalConfigurationEnvVarSource`	Value of the environment variable which will be passed to the Kafka Connect pods. It can be passed either as a reference to Secret or ConfigMap field. The field has to specify exactly one Secret or ConfigMap.

4.2.80. `ExternalConfigurationEnvVarSource` schema reference

Used in: `ExternalConfigurationEnv`

Property	Description
configMapKeyRef ConfigMapKeySelector	Reference to a key in a ConfigMap. For more information, see the external documentation for core/v1 configmapkeyselector.
secretKeyRef SecretKeySelector	Reference to a key in a Secret. For more information, see the external documentation for core/v1 secretkeyselector.

4.2.81. `ExternalConfigurationVolumeSource` schema reference

Used in: `ExternalConfiguration`

Property	Description
configMap ConfigMapVolumeSource	Reference to a key in a ConfigMap. Exactly one Secret or ConfigMap has to be specified. For more information, see the external documentation for core/v1 configmapvolumesource.

Property	Description
name	Name of the volume which will be added to the Kafka Connect pods.
string	
secret	Reference to a key in a Secret. Exactly one Secret or ConfigMap has to be specified. For more information, see the external documentation for core/v1 secretvolumesource.
SecretVolumeSource	

4.2.82. `Build` schema reference

Used in: `KafkaConnectSpec`

Full list of `Build` schema properties

Configures additional connectors for Kafka Connect deployments.

`output`

To build new container images with additional connector plugins, Strimzi requires a container registry where the images can be pushed to, stored, and pulled from. Strimzi does not run its own container registry, so a registry must be provided. Strimzi supports private container registries as well as public registries such as Quay or Docker Hub. The container registry is configured in the `.spec.build.output` section of the `KafkaConnect` custom resource. The `output` configuration, which is required, supports two types: `docker` and `imagestream`.

Using Docker registry

To use a Docker registry, you have to specify the `type` as `docker`, and the `image` field with the full name of the new container image. The full name must include:

- The address of the registry
- Port number (if listening on a non-standard port)
- The tag of the new container image

Example valid container image names:

- `docker.io/my-org/my-image/my-tag`
- `quay.io/my-org/my-image/my-tag`
- `image-registry.image-registry.svc:5000/myproject/kafka-connect-build:latest`

Each Kafka Connect deployment must use a separate image, which can mean different tags at the most basic level.

If the registry requires authentication, use the `pushSecret` to set a name of the Secret with the registry credentials. For the Secret, use the `kubernetes.io/dockerconfigjson` type and a `.dockerconfigjson` file to contain the Docker credentials. For more information on pulling an image from a private registry, see Create a Secret based on existing Docker credentials.

Example `output` configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      type: docker ①
      image: my-registry.io/my-org/my-connect-cluster:latest ②
      pushSecret: my-registry-credentials ③
  #...
```

① (Required) Type of output used by Strimzi.

② (Required) Full name of the image used, including the repository and tag.

③ (Optional) Name of the secret with the container registry credentials.

Using OpenShift ImageStream

Instead of Docker, you can use OpenShift ImageStream to store a new container image. The ImageStream has to be created manually before deploying Kafka Connect. To use ImageStream, set the `type` to `imagestream`, and use the `image` property to specify the name of the ImageStream and the tag used. For example, `my-connect-image-stream:latest`.

Example `output` configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      type: imagestream ①
      image: my-connect-build:latest ②
  #...
```

① (Required) Type of output used by Strimzi.

② (Required) Name of the ImageStream and tag.

`plugins`

Connector plugins are a set of files that define the implementation required to connect to certain types of external system. The connector plugins required for a container image must be configured using the `.spec.build.plugins` property of the `KafkaConnect` custom resource. Each connector plugin must have a name which is unique within the Kafka Connect deployment. Additionally, the plugin artifacts must be listed. These artifacts are downloaded by Strimzi, added to the new container

image, and used in the Kafka Connect deployment. The connector plugin artifacts can also include additional components, such as (de)serializers. Each connector plugin is downloaded into a separate directory so that the different connectors and their dependencies are properly *sandboxed*. Each plugin must be configured with at least one `artifact`.

Example `plugins` configuration with two connector plugins

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins: ①
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://repo1.maven.org/maven2/io/debezium/debezium-connector-
postgres/2.1.3.Final/debezium-connector-postgres-2.1.3.Final-plugin.tar.gz
            sha512sum:
c4ddc97846de561755dc0b021a62aba656098829c70eb3ade3b817ce06d852ca12ae50c0281cc791a5a131
cb7fc21fb15f4b8ee76c6cae5dd07f9c11cb7c6e79
      - name: camel-telegram
        artifacts:
          - type: tgz
            url:
https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-telegram-
kafka-connector/0.11.5/camel-telegram-kafka-connector-0.11.5-package.tar.gz
            sha512sum:
d6d9f45e0d1dbfcc9f6d1c7ca2046168c764389c78bc4b867dab32d24f710bb74ccf2a007d7d7a8af2dfca
09d9a52ccbc2831fc715c195a3634cca055185bd91
  #...
```

① (Required) List of connector plugins and their artifacts.

Strimzi supports the following types of artifacts:

- JAR files, which are downloaded and used directly
- TGZ archives, which are downloaded and unpacked
- ZIP archives, which are downloaded and unpacked
- Maven artifacts, which uses Maven coordinates
- Other artifacts, which are downloaded and used directly

IMPORTANT	Strimzi does not perform any security scanning of the downloaded artifacts. For security reasons, you should first verify the artifacts manually, and configure the checksum verification to make sure the same artifact is used in

the automated build and in the Kafka Connect deployment.

Using JAR artifacts

JAR artifacts represent a JAR file that is downloaded and added to a container image. To use a JAR artifacts, set the `type` property to `jar`, and specify the download location using the `url` property.

Additionally, you can specify a SHA-512 checksum of the artifact. If specified, Strimzi will verify the checksum of the artifact while building the new container image.

Example JAR artifact

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: jar ①
            url: https://my-domain.tld/my-jar.jar ②
            sha512sum: 589...ab4 ③
          - type: jar
            url: https://my-domain.tld/my-jar2.jar
  #...
```

① (Required) Type of artifact.

② (Required) URL from which the artifact is downloaded.

③ (Optional) SHA-512 checksum to verify the artifact.

Using TGZ artifacts

TGZ artifacts are used to download TAR archives that have been compressed using Gzip compression. The TGZ artifact can contain the whole Kafka Connect connector, even when comprising multiple different files. The TGZ artifact is automatically downloaded and unpacked by Strimzi while building the new container image. To use TGZ artifacts, set the `type` property to `tgz`, and specify the download location using the `url` property.

Additionally, you can specify a SHA-512 checksum of the artifact. If specified, Strimzi will verify the checksum before unpacking it and building the new container image.

Example TGZ artifact

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
```

```
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: tgz ①
            url: https://my-domain.tld/my-connector-archive.tgz ②
            sha512sum: 158...jg10 ③
  #...
```

① (Required) Type of artifact.

② (Required) URL from which the archive is downloaded.

③ (Optional) SHA-512 checksum to verify the artifact.

Using ZIP artifacts

ZIP artifacts are used to download ZIP compressed archives. Use ZIP artifacts in the same way as the TGZ artifacts described in the previous section. The only difference is you specify `type: zip` instead of `type: tgz`.

Using Maven artifacts

`maven` artifacts are used to specify connector plugin artifacts as Maven coordinates. The Maven coordinates identify plugin artifacts and dependencies so that they can be located and fetched from a Maven repository.

NOTE The Maven repository must be accessible for the connector build process to add the artifacts to the container image.

Example Maven artifact

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: maven ①
            repository: https://mvnrepository.com ②
            group: org.apache.camel.kafkaconnector ③
            artifact: camel-kafka-connector ④
```

```
          version: 0.11.0 ⑤
  #...
```

① (Required) Type of artifact.

② (Optional) Maven repository to download the artifacts from. If you do not specify a repository, Maven Central repository is used by default.

③ (Required) Maven group ID.

④ (Required) Maven artifact type.

⑤ (Required) Maven version number.

Using `other` artifacts

`other` artifacts represent any kind of file that is downloaded and added to a container image. If you want to use a specific name for the artifact in the resulting container image, use the `fileName` field. If a file name is not specified, the file is named based on the URL hash.

Additionally, you can specify a SHA-512 checksum of the artifact. If specified, Strimzi will verify the checksum of the artifact while building the new container image.

Example `other` artifact

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: other  ①
            url: https://my-domain.tld/my-other-file.ext  ②
            sha512sum: 589...ab4  ③
            fileName: name-the-file.ext  ④
  #...
```

① (Required) Type of artifact.

② (Required) URL from which the artifact is downloaded.

③ (Optional) SHA-512 checksum to verify the artifact.

④ (Optional) The name under which the file is stored in the resulting container image.

`Build` schema properties

Property	Description
output DockerOutput, ImageStreamOutput	Configures where should the newly built image be stored. Required. The type depends on the value of the output.type property within the given object, which must be one of [docker, imagestream].
resources ResourceRequirements	CPU and memory resources to reserve for the build. For more information, see the external documentation for core/v1 resourcerequirements.
plugins Plugin array	List of connector plugins which should be added to the Kafka Connect. Required.

4.2.83. DockerOutput schema reference

Used in: Build

The type property is a discriminator that distinguishes use of the DockerOutput type from ImageStreamOutput. It must have the value docker for the type DockerOutput.

Property	Description
image string	The full name which should be used for tagging and pushing the newly built image. For example quay.io/my-organization/my-custom-connect:latest. Required.
pushSecret string	Container Registry Secret with the credentials for pushing the newly built image.
additionalKanikoOptions string array	Configures additional options which will be passed to the Kaniko executor when building the new Connect image. Allowed options are: --customPlatform, --insecure, --insecure-pull, --insecure-registry, --log-format, --log-timestamp, --registry-mirror, --reproducible, --single-snapshot, --skip-tls-verify, --skip-tls-verify-pull, --skip-tls-verify-registry, --verbosity, --snapshotMode, --use-new-run. These options will be used only on Kubernetes where the Kaniko executor is used. They will be ignored on OpenShift. The options are described in the Kaniko GitHub repository. Changing this field does not trigger new build of the Kafka Connect image.
type string	Must be docker.

4.2.84. `ImageStreamOutput` schema reference

Used in: `Build`

The `type` property is a discriminator that distinguishes use of the `ImageStreamOutput` type from `DockerOutput`. It must have the value `imagestream` for the type `ImageStreamOutput`.

Property	Description
image string	The name and tag of the ImageStream where the newly built image will be pushed. For example `my-custom-connect:latest`. Required.
type string	Must be `imagestream`.

4.2.85. `Plugin` schema reference

Used in: `Build`

Property	Description
name string	The unique name of the connector plugin. Will be used to generate the path where the connector artifacts will be stored. The name has to be unique within the KafkaConnect resource. The name has to follow the following pattern: `^[a-z][-_a-z0-9]*[a-z]$`. Required.
artifacts `JarArtifact`, `TgzArtifact`, `ZipArtifact`, `MavenArtifact`, `OtherArtifact` array	List of artifacts which belong to this connector plugin. Required.

4.2.86. `JarArtifact` schema reference

Used in: `Plugin`

Property	Description
url string	URL of the artifact which will be downloaded. Strimzi does not do any security scanning of the downloaded artifacts. For security reasons, you should first verify the artifacts manually and configure the checksum verification to make sure the same artifact is used in the automated build. Required for `jar`, `zip`, `tgz` and `other` artifacts. Not applicable to the `maven` artifact type.

Property	Description
sha512sum	SHA512 checksum of the artifact. Optional. If specified, the checksum will be verified while building the new container. If not specified, the downloaded artifact will not be verified. Not applicable to the `maven` artifact type.
string	
insecure	By default, connections using TLS are verified to check they are secure. The server certificate used must be valid, trusted, and contain the server name. By setting this option to `true`, all TLS verification is disabled and the artifact will be downloaded, even when the server is considered insecure.
boolean	
type	Must be `jar`.
string	

4.2.87. `TgzArtifact` schema reference

Used in: `Plugin`

Property	Description
url	URL of the artifact which will be downloaded. Strimzi does not do any security scanning of the downloaded artifacts. For security reasons, you should first verify the artifacts manually and configure the checksum verification to make sure the same artifact is used in the automated build. Required for `jar`, `zip`, `tgz` and `other` artifacts. Not applicable to the `maven` artifact type.
string	
sha512sum	SHA512 checksum of the artifact. Optional. If specified, the checksum will be verified while building the new container. If not specified, the downloaded artifact will not be verified. Not applicable to the `maven` artifact type.
string	
insecure	By default, connections using TLS are verified to check they are secure. The server certificate used must be valid, trusted, and contain the server name. By setting this option to `true`, all TLS verification is disabled and the artifact will be downloaded, even when the server is considered insecure.
boolean	
type	Must be `tgz`.
string	

4.2.88. `ZipArtifact` schema reference

Used in: `Plugin`

Property	Description
url	URL of the artifact which will be downloaded. Strimzi does not do any security scanning of the downloaded artifacts. For security reasons, you should first verify the artifacts manually and configure the checksum verification to make sure the same artifact is used in the automated build. Required for `jar`, `zip`, `tgz` and `other` artifacts. Not applicable to the `maven` artifact type.
string	
sha512sum	SHA512 checksum of the artifact. Optional. If specified, the checksum will be verified while building the new container. If not specified, the downloaded artifact will not be verified. Not applicable to the `maven` artifact type.
string	
insecure	By default, connections using TLS are verified to check they are secure. The server certificate used must be valid, trusted, and contain the server name. By setting this option to `true`, all TLS verification is disabled and the artifact will be downloaded, even when the server is considered insecure.
boolean	
type	Must be `zip`.
string	

4.2.89. `MavenArtifact` schema reference

Used in: `Plugin`

The `type` property is a discriminator that distinguishes use of the `MavenArtifact` type from `JarArtifact`, `TgzArtifact`, `ZipArtifact`, `OtherArtifact`. It must have the value `maven` for the type `MavenArtifact`.

Property	Description
repository	Maven repository to download the artifact from. Applicable to the `maven` artifact type only.
string	
group	Maven group id. Applicable to the `maven` artifact type only.
string	
artifact	Maven artifact id. Applicable to the `maven` artifact type only.
string	

Property	Description
version	Maven version number. Applicable to the `maven` artifact type only.
string	
type	Must be `maven`.
string	

4.2.90. `OtherArtifact` schema reference

Used in: `Plugin`

Property	Description
url	URL of the artifact which will be downloaded. Strimzi does not do any security scanning of the downloaded artifacts. For security reasons, you should first verify the artifacts manually and configure the checksum verification to make sure the same artifact is used in the automated build. Required for `jar`, `zip`, `tgz` and `other` artifacts. Not applicable to the `maven` artifact type.
string	
sha512sum	SHA512 checksum of the artifact. Optional. If specified, the checksum will be verified while building the new container. If not specified, the downloaded artifact will not be verified. Not applicable to the `maven` artifact type.
string	
fileName	Name under which the artifact will be stored.
string	
insecure	By default, connections using TLS are verified to check they are secure. The server certificate used must be valid, trusted, and contain the server name. By setting this option to `true`, all TLS verification is disabled and the artifact will be downloaded, even when the server is considered insecure.
boolean	
type	Must be `other`.
string	

4.2.91. `KafkaConnectStatus` schema reference

Used in: `KafkaConnect`

Property	Description
conditions	List of status conditions.
`Condition` array	
observedGeneration	The generation of the CRD that was last reconciled by the operator.
integer	
url	The URL of the REST API endpoint for managing and monitoring Kafka Connect connectors.
string	
connectorPlugins	The list of connector plugins available in this Kafka Connect deployment.
`ConnectorPlugin` array	
labelSelector	Label selector for pods providing this resource.
string	
replicas	The current number of pods being used to provide this resource.
integer	

4.2.92. `ConnectorPlugin` schema reference

Used in: `KafkaConnectStatus`, `KafkaMirrorMaker2Status`

Property	Description
type	The type of the connector plugin. The available types are `sink` and `source`.
string	
version	The version of the connector plugin.
string	
class	The class of the connector plugin.
string	

4.2.93. `KafkaTopic` schema reference

Property	Description
spec	The specification of the topic.
`KafkaTopicSpec`	
status	The status of the topic.
`KafkaTopicStatus`	

4.2.94. `KafkaTopicSpec` schema reference

Used in: `KafkaTopic`

Property	Description
partitions integer	The number of partitions the topic should have. This cannot be decreased after topic creation. It can be increased after topic creation, but it is important to understand the consequences that has, especially for topics with semantic partitioning. When absent this will default to the broker configuration for `num.partitions`.
replicas integer	The number of replicas the topic should have. When absent this will default to the broker configuration for `default.replication.factor`.
config map	The topic configuration.
topicName string	The name of the topic. When absent this will default to the metadata.name of the topic. It is recommended to not set this unless the topic name is not a valid Kubernetes resource name.

4.2.95. `KafkaTopicStatus` schema reference

Used in: `KafkaTopic`

Property	Description
conditions `Condition` array	List of status conditions.
observedGeneration integer	The generation of the CRD that was last reconciled by the operator.
topicName string	Topic name.

4.2.96. `KafkaUser` schema reference

Property	Description
spec `KafkaUserSpec`	The specification of the user.
status `KafkaUserStatus`	The status of the Kafka User.

4.2.97. `KafkaUserSpec` schema reference

Used in: `KafkaUser`

Property	Description
authentication	Authentication mechanism enabled for this Kafka user. The supported authentication mechanisms are `scram-sha-512`, `tls`, and `tls-external`. • `scram-sha-512` generates a secret with SASL SCRAM-SHA-512 credentials. • `tls` generates a secret with user certificate for mutual TLS authentication. • `tls-external` does not generate a user certificate. But prepares the user for using mutual TLS authentication using a user certificate generated outside the User Operator. ACLs and quotas set for this user are configured in the `CN=<username>` format. Authentication is optional. If authentication is not configured, no credentials are generated. ACLs and quotas set for the user are configured in the `<username>` format suitable for SASL authentication. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, tls-external, scram-sha-512].
`KafkaUserTlsClientAuthentication`, `KafkaUserTlsExternalClientAuthentication`, `KafkaUserScramSha512ClientAuthentication`	
authorization	Authorization rules for this Kafka user. The type depends on the value of the `authorization.type` property within the given object, which must be one of [simple].
`KafkaUserAuthorizationSimple`	
quotas	Quotas on requests to control the broker resources used by clients. Network bandwidth and request rate quotas can be enforced.Kafka documentation for Kafka User quotas can be found at http://kafka.apache.org/documentation/#design_quotas.
`KafkaUserQuotas`	
template	Template to specify how Kafka User `Secrets` are generated.
`KafkaUserTemplate`	

4.2.98. `KafkaUserTlsClientAuthentication` schema reference

Used in: `KafkaUserSpec`

The `type` property is a discriminator that distinguishes use of the `KafkaUserTlsClientAuthentication` type from `KafkaUserTlsExternalClientAuthentication`, `KafkaUserScramSha512ClientAuthentication`. It must have the value `tls` for the type `KafkaUserTlsClientAuthentication`.

Property	Description
type	Must be `tls`.
string	

4.2.99. `KafkaUserTlsExternalClientAuthentication` schema reference

Used in: `KafkaUserSpec`

The `type` property is a discriminator that distinguishes use of the `KafkaUserTlsExternalClientAuthentication` type from `KafkaUserTlsClientAuthentication`, `KafkaUserScramSha512ClientAuthentication`. It must have the value `tls-external` for the type `KafkaUserTlsExternalClientAuthentication`.

Property	Description
type	Must be `tls-external`.
string	

4.2.100. `KafkaUserScramSha512ClientAuthentication` schema reference

Used in: `KafkaUserSpec`

The `type` property is a discriminator that distinguishes use of the `KafkaUserScramSha512ClientAuthentication` type from `KafkaUserTlsClientAuthentication`, `KafkaUserTlsExternalClientAuthentication`. It must have the value `scram-sha-512` for the type `KafkaUserScramSha512ClientAuthentication`.

Property	Description
password	Specify the password for the user. If not set, a new password is generated by the User Operator.
`Password`	
type	Must be `scram-sha-512`.
string	

4.2.101. `Password` schema reference

Used in: `KafkaUserScramSha512ClientAuthentication`

Property	Description
valueFrom	Secret from which the password should be read.
`PasswordSource`	

4.2.102. `PasswordSource` schema reference

Used in: `Password`

Property	Description
secretKeyRef	Selects a key of a Secret in the resource's namespace. For more information, see the external documentation for core/v1 secretkeyselector.
SecretKeySelector	

4.2.103. `KafkaUserAuthorizationSimple` schema reference

Used in: `KafkaUserSpec`

The `type` property is a discriminator that distinguishes use of the `KafkaUserAuthorizationSimple` type from other subtypes which may be added in the future. It must have the value `simple` for the type `KafkaUserAuthorizationSimple`.

Property	Description
type	Must be `simple`.
string	
acls	List of ACL rules which should be applied to this user.
`AclRule` array	

4.2.104. `AclRule` schema reference

Used in: `KafkaUserAuthorizationSimple`

Full list of `AclRule` schema properties

Configures access control rules for a `KafkaUser` when brokers are using the `AclAuthorizer`.

Example `KafkaUser` configuration with authorization

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  # ...
  authorization:
    type: simple
    acls:
      - resource:
          type: topic
          name: my-topic
          patternType: literal
        operations:
          - Read
          - Describe
```

```
    - resource:
        type: group
        name: my-group
        patternType: prefix
      operations:
        - Read
```

`resource`

Use the `resource` property to specify the resource that the rule applies to.

Simple authorization supports four resource types, which are specified in the `type` property:

- Topics (`topic`)
- Consumer Groups (`group`)
- Clusters (`cluster`)
- Transactional IDs (`transactionalId`)

For Topic, Group, and Transactional ID resources you can specify the name of the resource the rule applies to in the `name` property.

Cluster type resources have no name.

A name is specified as a `literal` or a `prefix` using the `patternType` property.

- Literal names are taken exactly as they are specified in the `name` field.
- Prefix names use the `name` value as a prefix and then apply the rule to all resources with names starting with that value.

When `patternType` is set as `literal`, you can set the name to `*` to indicate that the rule applies to all resources.

Example ACL rule that allows the user to read messages from all topics

```
  acls:
    - resource:
        type: topic
        name: "*"
        patternType: literal
      operations:
        - Read
```

`type`

The `type` of rule, which is to `allow` or `deny` (not currently supported) an operations.

The `type` field is optional. If `type` is unspecified, the ACL rule is treated as an `allow` rule.

`operations`

Specify a list of `operations` for the rule to allow or deny.

The following operations are supported:

- Read
- Write
- Delete
- Alter
- Describe
- All
- IdempotentWrite
- ClusterAction
- Create
- AlterConfigs
- DescribeConfigs

Only certain operations work with each resource.

For more details about `AclAuthorizer`, ACLs and supported combinations of resources and operations, see Authorization and ACLs.

`host`

Use the `host` property to specify a remote host from which the rule is allowed or denied.

Use an asterisk (*) to allow or deny the operation from all hosts. The `host` field is optional. If `host` is unspecified, the `*` value is used by default.

`AclRule` schema properties

Property	Description
host string	The host from which the action described in the ACL rule is allowed or denied.
operation	**The `operation` property has been deprecated, and should now be configured using `spec.authorization.acls[*].operations`.** Operation which will be allowed or denied. Supported operations are: Read, Write, Create, Delete, Alter, Describe, ClusterAction, AlterConfigs, DescribeConfigs, IdempotentWrite and All.
string (one of [Read, Write, Delete, Alter, Describe, All, IdempotentWrite, ClusterAction, Create, AlterConfigs, DescribeConfigs])	

Property	Description
operations string (one or more of [Read, Write, Delete, Alter, Describe, All, IdempotentWrite, ClusterAction, Create, AlterConfigs, DescribeConfigs]) array	List of operations which will be allowed or denied. Supported operations are: Read, Write, Create, Delete, Alter, Describe, ClusterAction, AlterConfigs, DescribeConfigs, IdempotentWrite and All.
resource `AclRuleTopicResource`, `AclRuleGroupResource`, `AclRuleClusterResource`, `AclRuleTransactionalIdResource`	Indicates the resource for which given ACL rule applies. The type depends on the value of the `resource.type` property within the given object, which must be one of [topic, group, cluster, transactionalId].
type string (one of [allow, deny])	The type of the rule. Currently the only supported type is `allow`. ACL rules with type `allow` are used to allow user to execute the specified operations. Default value is `allow`.

4.2.105. `AclRuleTopicResource` schema reference

Used in: `AclRule`

The `type` property is a discriminator that distinguishes use of the `AclRuleTopicResource` type from `AclRuleGroupResource`, `AclRuleClusterResource`, `AclRuleTransactionalIdResource`. It must have the value `topic` for the type `AclRuleTopicResource`.

Property	Description
type string	Must be `topic`.
name string	Name of resource for which given ACL rule applies. Can be combined with `patternType` field to use prefix pattern.
patternType string (one of [prefix, literal])	Describes the pattern used in the resource field. The supported types are `literal` and `prefix`. With `literal` pattern type, the resource field will be used as a definition of a full topic name. With `prefix` pattern type, the resource name will be used only as a prefix. Default value is `literal`.

4.2.106. `AclRuleGroupResource` schema reference

Used in: `AclRule`

The `type` property is a discriminator that distinguishes use of the `AclRuleGroupResource` type from `AclRuleTopicResource`, `AclRuleClusterResource`, `AclRuleTransactionalIdResource`. It must have the

value `group` for the type `AclRuleGroupResource`.

Property	Description
type	Must be `group`.
string	
name	Name of resource for which given ACL rule applies. Can be combined with `patternType` field to use prefix pattern.
string	
patternType	Describes the pattern used in the resource field. The supported types are `literal` and `prefix`. With `literal` pattern type, the resource field will be used as a definition of a full topic name. With `prefix` pattern type, the resource name will be used only as a prefix. Default value is `literal`.
string (one of [prefix, literal])	

4.2.107. `AclRuleClusterResource` schema reference

Used in: `AclRule`

The `type` property is a discriminator that distinguishes use of the `AclRuleClusterResource` type from `AclRuleTopicResource`, `AclRuleGroupResource`, `AclRuleTransactionalIdResource`. It must have the value `cluster` for the type `AclRuleClusterResource`.

Property	Description
type	Must be `cluster`.
string	

4.2.108. `AclRuleTransactionalIdResource` schema reference

Used in: `AclRule`

The `type` property is a discriminator that distinguishes use of the `AclRuleTransactionalIdResource` type from `AclRuleTopicResource`, `AclRuleGroupResource`, `AclRuleClusterResource`. It must have the value `transactionalId` for the type `AclRuleTransactionalIdResource`.

Property	Description
type	Must be `transactionalId`.
string	
name	Name of resource for which given ACL rule applies. Can be combined with `patternType` field to use prefix pattern.
string	

Property	Description
patternType	Describes the pattern used in the resource field. The supported types are `literal` and `prefix`. With `literal` pattern type, the resource field will be used as a definition of a full name. With `prefix` pattern type, the resource name will be used only as a prefix. Default value is `literal`.
string (one of [prefix, literal])	

4.2.109. `KafkaUserQuotas` schema reference

Used in: `KafkaUserSpec`

Full list of `KafkaUserQuotas` schema properties

Kafka allows a user to set `quotas` to control the use of resources by clients.

`quotas`

You can configure your clients to use the following types of quotas:

- *Network usage* quotas specify the byte rate threshold for each group of clients sharing a quota.
- *CPU utilization* quotas specify a window for broker requests from clients. The window is the percentage of time for clients to make requests. A client makes requests on the I/O threads and network threads of the broker.
- *Partition mutation* quotas limit the number of partition mutations which clients are allowed to make per second.

A partition mutation quota prevents Kafka clusters from being overwhelmed by concurrent topic operations. Partition mutations occur in response to the following types of user requests:

- Creating partitions for a new topic
- Adding partitions to an existing topic
- Deleting partitions from a topic

You can configure a partition mutation quota to control the rate at which mutations are accepted for user requests.

Using quotas for Kafka clients might be useful in a number of situations. Consider a wrongly configured Kafka producer which is sending requests at too high a rate. Such misconfiguration can cause a denial of service to other clients, so the problematic client ought to be blocked. By using a network limiting quota, it is possible to prevent this situation from significantly impacting other clients.

Strimzi supports user-level quotas, but not client-level quotas.

Example Kafka user quota configuration

```
spec:
```

```
  quotas:
    producerByteRate: 1048576
    consumerByteRate: 2097152
    requestPercentage: 55
    controllerMutationRate: 10
```

For more information about Kafka user quotas, refer to the Apache Kafka documentation.

`KafkaUserQuotas` schema properties

Property	Description
consumerByteRate integer	A quota on the maximum bytes per-second that each client group can fetch from a broker before the clients in the group are throttled. Defined on a per-broker basis.
controllerMutationRate number	A quota on the rate at which mutations are accepted for the create topics request, the create partitions request and the delete topics request. The rate is accumulated by the number of partitions created or deleted.
producerByteRate integer	A quota on the maximum bytes per-second that each client group can publish to a broker before the clients in the group are throttled. Defined on a per-broker basis.
requestPercentage integer	A quota on the maximum CPU utilization of each client group as a percentage of network and I/O threads.

4.2.110. `KafkaUserTemplate` schema reference

Used in: `KafkaUserSpec`

Full list of `KafkaUserTemplate` schema properties

Specify additional labels and annotations for the secret created by the User Operator.

An example showing the `KafkaUserTemplate`

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  template:
```

```
    secret:
      metadata:
        labels:
          label1: value1
        annotations:
          anno1: value1
  # ...
```

`KafkaUserTemplate` **schema properties**

Property	Description
secret `ResourceTemplate`	Template for KafkaUser resources. The template allows users to specify how the `Secret` with password or TLS certificates is generated.

4.2.111. `KafkaUserStatus` schema reference

Used in: `KafkaUser`

Property	Description
conditions `Condition` array	List of status conditions.
observedGeneration integer	The generation of the CRD that was last reconciled by the operator.
username string	Username.
secret string	The name of `Secret` where the credentials are stored.

4.2.112. `KafkaMirrorMaker` schema reference

The type `KafkaMirrorMaker` has been deprecated. Please use `KafkaMirrorMaker2` instead.

Property	Description
spec `KafkaMirrorMakerSpec`	The specification of Kafka MirrorMaker.
status `KafkaMirrorMakerStatus`	The status of Kafka MirrorMaker.

4.2.113. `KafkaMirrorMakerSpec` schema reference

Used in: `KafkaMirrorMaker`

Full list of `KafkaMirrorMakerSpec` schema properties

Configures Kafka MirrorMaker.

`include`

Use the `include` property to configure a list of topics that Kafka MirrorMaker mirrors from the source to the target Kafka cluster.

The property allows any regular expression from the simplest case with a single topic name to complex patterns. For example, you can mirror topics A and B using `A|B` or all topics using `*`. You can also pass multiple regular expressions separated by commas to the Kafka MirrorMaker.

`KafkaMirrorMakerConsumerSpec` and `KafkaMirrorMakerProducerSpec`

Use the `KafkaMirrorMakerConsumerSpec` and `KafkaMirrorMakerProducerSpec` to configure source (consumer) and target (producer) clusters.

Kafka MirrorMaker always works together with two Kafka clusters (source and target). To establish a connection, the bootstrap servers for the source and the target Kafka clusters are specified as comma-separated lists of `HOSTNAME:PORT` pairs. Each comma-separated list contains one or more Kafka brokers or a `Service` pointing to Kafka brokers specified as a `HOSTNAME:PORT` pair.

`logging`

Kafka MirrorMaker has its own configurable logger:

- `mirrormaker.root.logger`

MirrorMaker uses the Apache `log4j` logger implementation.

Use the `logging` property to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. Inside the ConfigMap, the logging configuration is described using `log4j.properties`. Both `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. A ConfigMap using the exact logging configuration specified is created with the custom resource when the Cluster Operator is running, then recreated after each reconciliation. If you do not specify a custom ConfigMap, default logging settings are used. If a specific logger value is not set, upper-level logger settings are inherited for that logger. For more information about log levels, see Apache logging services.

Here we see examples of `inline` and `external` logging:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
spec:
  # ...
  logging:
```

```
    type: inline
    loggers:
      mirrormaker.root.logger: "INFO"
  # ...
```

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: customConfigMap
        key: mirror-maker-log4j.properties
  # ...
```

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`KafkaMirrorMakerSpec` schema properties

Property	Description
version	The Kafka MirrorMaker version. Defaults to 3.4.0. Consult the documentation to understand the process required to upgrade or downgrade the version.
string	
replicas	The number of pods in the `Deployment`.
integer	
image	The docker image for the pods.
string	
consumer	Configuration of source cluster.
`KafkaMirrorMakerConsumerSpec`	
producer	Configuration of target cluster.
`KafkaMirrorMakerProducerSpec`	
resources	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	

Property	Description
whitelist	**The `whitelist` property has been deprecated, and should now be configured using `spec.include`.** List of topics which are included for mirroring. This option allows any regular expression using Java-style regular expressions. Mirroring two topics named A and B is achieved by using the expression `A|B`. Or, as a special case, you can mirror all topics using the regular expression `*`. You can also specify multiple regular expressions separated by commas.
string	
include	List of topics which are included for mirroring. This option allows any regular expression using Java-style regular expressions. Mirroring two topics named A and B is achieved by using the expression `A|B`. Or, as a special case, you can mirror all topics using the regular expression `*`. You can also specify multiple regular expressions separated by commas.
string	
jvmOptions	JVM Options for pods.
`JvmOptions`	
logging	Logging configuration for MirrorMaker. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
`InlineLogging`, `ExternalLogging`	
metricsConfig	Metrics configuration. The type depends on the value of the `metricsConfig.type` property within the given object, which must be one of [jmxPrometheusExporter].
`JmxPrometheusExporterMetrics`	
tracing	The configuration of tracing in Kafka MirrorMaker. The type depends on the value of the `tracing.type` property within the given object, which must be one of [jaeger, opentelemetry].
`JaegerTracing`, `OpenTelemetryTracing`	
template	Template to specify how Kafka MirrorMaker resources, `Deployments` and `Pods`, are generated.
`KafkaMirrorMakerTemplate`	
livenessProbe	Pod liveness checking.
`Probe`	
readinessProbe	Pod readiness checking.
`Probe`	

4.2.114. KafkaMirrorMakerConsumerSpec schema reference

Used in: KafkaMirrorMakerSpec

Full list of KafkaMirrorMakerConsumerSpec schema properties

Configures a MirrorMaker consumer.

numStreams

Use the consumer.numStreams property to configure the number of streams for the consumer.

You can increase the throughput in mirroring topics by increasing the number of consumer threads. Consumer threads belong to the consumer group specified for Kafka MirrorMaker. Topic partitions are assigned across the consumer threads, which consume messages in parallel.

offsetCommitInterval

Use the consumer.offsetCommitInterval property to configure an offset auto-commit interval for the consumer.

You can specify the regular time interval at which an offset is committed after Kafka MirrorMaker has consumed data from the source Kafka cluster. The time interval is set in milliseconds, with a default value of 60,000.

config

Use the consumer.config properties to configure Kafka options for the consumer as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the Apache Kafka configuration documentation for consumers.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Kafka cluster bootstrap address
- Security (encryption, authentication, and authorization)
- Consumer group identifier
- Interceptors

Properties with the following prefixes cannot be set:

- `bootstrap.servers`
- `group.id`
- `interceptor.classes`
- `sasl.`
- `security.`
- `ssl.`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to MirrorMaker, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites

IMPORTANT	The Cluster Operator does not validate keys or values in the `config` object provided. If an invalid configuration is provided, the MirrorMaker cluster might not start or might become unstable. In this case, fix the configuration so that the Cluster Operator can roll out the new configuration to all MirrorMaker nodes.

`groupId`

Use the `consumer.groupId` property to configure a consumer group identifier for the consumer.

Kafka MirrorMaker uses a Kafka consumer to consume messages, behaving like any other Kafka consumer client. Messages consumed from the source Kafka cluster are mirrored to a target Kafka cluster. A group identifier is required, as the consumer needs to be part of a consumer group for the assignment of partitions.

`KafkaMirrorMakerConsumerSpec` schema properties

Property	Description
numStreams	Specifies the number of consumer stream threads to create.
integer	
offsetCommitInterval	Specifies the offset auto-commit interval in ms. Default value is 60000.
integer	
bootstrapServers	A list of host:port pairs for establishing the initial connection to the Kafka cluster.
string	
groupId	A unique string that identifies the consumer group this consumer belongs to.
string	

Property	Description
authentication	Authentication configuration for connecting to the cluster. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, scram-sha-256, scram-sha-512, plain, oauth].
`KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationOAuth`	
config	The MirrorMaker consumer config. Properties with the following prefixes cannot be set: ssl., bootstrap.servers, group.id, sasl., security., interceptor.classes (with the exception of: ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols).
map	
tls	TLS configuration for connecting MirrorMaker to the cluster.
`ClientTls`	

4.2.115. `KafkaMirrorMakerProducerSpec` schema reference

Used in: `KafkaMirrorMakerSpec`

Full list of `KafkaMirrorMakerProducerSpec` schema properties

Configures a MirrorMaker producer.

`abortOnSendFailure`

Use the `producer.abortOnSendFailure` property to configure how to handle message send failure from the producer.

By default, if an error occurs when sending a message from Kafka MirrorMaker to a Kafka cluster:

- The Kafka MirrorMaker container is terminated in Kubernetes.
- The container is then recreated.

If the `abortOnSendFailure` option is set to `false`, message sending errors are ignored.

`config`

Use the `producer.config` properties to configure Kafka options for the producer as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the Apache Kafka configuration documentation for producers.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Kafka cluster bootstrap address
- Security (encryption, authentication, and authorization)
- Interceptors

Properties with the following prefixes cannot be set:

- `bootstrap.servers`
- `interceptor.classes`
- `sasl.`
- `security.`
- `ssl.`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to MirrorMaker, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites

IMPORTANT	The Cluster Operator does not validate keys or values in the `config` object provided. If an invalid configuration is provided, the MirrorMaker cluster might not start or might become unstable. In this case, fix the configuration so that the Cluster Operator can roll out the new configuration to all MirrorMaker nodes.

`KafkaMirrorMakerProducerSpec` schema properties

Property	Description
bootstrapServers string	A list of host:port pairs for establishing the initial connection to the Kafka cluster.
abortOnSendFailure boolean	Flag to set the MirrorMaker to exit on a failed send. Default value is `true`.

Property	Description
authentication `KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationOAuth`	Authentication configuration for connecting to the cluster. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, scram-sha-256, scram-sha-512, plain, oauth].
config map	The MirrorMaker producer config. Properties with the following prefixes cannot be set: ssl., bootstrap.servers, sasl., security., interceptor.classes (with the exception of: ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols).
tls `ClientTls`	TLS configuration for connecting MirrorMaker to the cluster.

4.2.116. `KafkaMirrorMakerTemplate` schema reference

Used in: `KafkaMirrorMakerSpec`

Property	Description
deployment `DeploymentTemplate`	Template for Kafka MirrorMaker `Deployment`.
pod `PodTemplate`	Template for Kafka MirrorMaker `Pods`.
podDisruptionBudget `PodDisruptionBudgetTemplate`	Template for Kafka MirrorMaker `PodDisruptionBudget`.
mirrorMakerContainer `ContainerTemplate`	Template for Kafka MirrorMaker container.
serviceAccount `ResourceTemplate`	Template for the Kafka MirrorMaker service account.

4.2.117. `KafkaMirrorMakerStatus` schema reference

Used in: `KafkaMirrorMaker`

Property	Description
conditions `Condition` array	List of status conditions.

Property	Description
observedGeneration	The generation of the CRD that was last reconciled by the operator.
integer	
labelSelector	Label selector for pods providing this resource.
string	
replicas	The current number of pods being used to provide this resource.
integer	

4.2.118. `KafkaBridge` schema reference

Property	Description
spec	The specification of the Kafka Bridge.
`KafkaBridgeSpec`	
status	The status of the Kafka Bridge.
`KafkaBridgeStatus`	

4.2.119. `KafkaBridgeSpec` schema reference

Used in: `KafkaBridge`

Full list of `KafkaBridgeSpec` schema properties

Configures a Kafka Bridge cluster.

Configuration options relate to:

- Kafka cluster bootstrap address
- Security (encryption, authentication, and authorization)
- Consumer configuration
- Producer configuration
- HTTP configuration

`logging`

Kafka Bridge has its own configurable loggers:

- `logger.bridge`
- `logger.<operation-id>`

You can replace `<operation-id>` in the `logger.<operation-id>` logger to set log levels for specific operations:

- `createConsumer`
- `deleteConsumer`

- subscribe
- unsubscribe
- poll
- assign
- commit
- send
- sendToPartition
- seekToBeginning
- seekToEnd
- seek
- healthy
- ready
- openapi

Each operation is defined according OpenAPI specification, and has a corresponding API endpoint through which the bridge receives requests from HTTP clients. You can change the log level on each endpoint to create fine-grained logging information about the incoming and outgoing HTTP requests.

Each logger has to be configured assigning it a `name` as `http.openapi.operation.<operation-id>`. For example, configuring the logging level for the `send` operation logger means defining the following:

```
logger.send.name = http.openapi.operation.send
logger.send.level = DEBUG
```

Kafka Bridge uses the Apache `log4j2` logger implementation. Loggers are defined in the `log4j2.properties` file, which has the following default configuration for `healthy` and `ready` endpoints:

```
logger.healthy.name = http.openapi.operation.healthy
logger.healthy.level = WARN
logger.ready.name = http.openapi.operation.ready
logger.ready.level = WARN
```

The log level of all other operations is set to `INFO` by default.

Use the `logging` property to configure loggers and logger levels.

You can set the log levels by specifying the logger and level directly (inline) or use a custom (external) ConfigMap. If a ConfigMap is used, you set `logging.valueFrom.configMapKeyRef.name` property to the name of the ConfigMap containing the external logging configuration. The `logging.valueFrom.configMapKeyRef.name` and `logging.valueFrom.configMapKeyRef.key` properties are mandatory. Default logging is used if the `name` or `key` is not set. Inside the ConfigMap, the logging

configuration is described using `log4j.properties`. For more information about log levels, see Apache logging services.

Here we see examples of `inline` and `external` logging.

Inline logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
spec:
  # ...
  logging:
    type: inline
    loggers:
      logger.bridge.level: "INFO"
      # enabling DEBUG just for send operation
      logger.send.name: "http.openapi.operation.send"
      logger.send.level: "DEBUG"
  # ...
```

External logging

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
spec:
  # ...
  logging:
    type: external
    valueFrom:
      configMapKeyRef:
        name: customConfigMap
        key: bridge-logj42.properties
  # ...
```

Any available loggers that are not configured have their level set to `OFF`.

If the Kafka Bridge was deployed using the Cluster Operator, changes to Kafka Bridge logging levels are applied dynamically.

If you use external logging, a rolling update is triggered when logging appenders are changed.

Garbage collector (GC)

Garbage collector logging can also be enabled (or disabled) using the `jvmOptions` property.

`KafkaBridgeSpec` schema properties

Property	Description
replicas	The number of pods in the `Deployment`. Defaults to `1`.
integer	

Property	Description
image	The docker image for the pods.
string	
bootstrapServers	A list of host:port pairs for establishing the initial connection to the Kafka cluster.
string	
tls	TLS configuration for connecting Kafka Bridge to the cluster.
`ClientTls`	
authentication	Authentication configuration for connecting to the cluster. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, scram-sha-256, scram-sha-512, plain, oauth].
`KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationOAuth`	
http	The HTTP related configuration.
`KafkaBridgeHttpConfig`	
adminClient	Kafka AdminClient related configuration.
`KafkaBridgeAdminClientSpec`	
consumer	Kafka consumer related configuration.
`KafkaBridgeConsumerSpec`	
producer	Kafka producer related configuration.
`KafkaBridgeProducerSpec`	
resources	CPU and memory resources to reserve. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
jvmOptions	**Currently not supported** JVM Options for pods.
`JvmOptions`	
logging	Logging configuration for Kafka Bridge. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
`InlineLogging`, `ExternalLogging`	
clientRackInitImage	The image of the init container used for initializing the `client.rack`.
string	
rack	Configuration of the node label which will be used as the client.rack consumer configuration.
`Rack`	
enableMetrics	Enable the metrics for the Kafka Bridge. Default is false.
boolean	

Property	Description
livenessProbe	Pod liveness checking.
Probe	
readinessProbe	Pod readiness checking.
Probe	
template	Template for Kafka Bridge resources. The template allows users to specify how a Deployment and Pod is generated.
KafkaBridgeTemplate	
tracing	The configuration of tracing in Kafka Bridge. The type depends on the value of the tracing.type property within the given object, which must be one of [jaeger, opentelemetry].
JaegerTracing, OpenTelemetryTracing	

4.2.120. KafkaBridgeHttpConfig schema reference

Used in: KafkaBridgeSpec

Full list of KafkaBridgeHttpConfig schema properties

Configures HTTP access to a Kafka cluster for the Kafka Bridge.

The default HTTP configuration is for the Kafka Bridge to listen on port 8080.

cors

As well as enabling HTTP access to a Kafka cluster, HTTP properties provide the capability to 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 access methods. For the origins, you can use a URL or a Java regular expression.

Example Kafka Bridge HTTP configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  http:
    port: 8080
    cors:
      allowedOrigins: "https://strimzi.io"
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  # ...
```

`KafkaBridgeHttpConfig` **schema properties**

Property	Description
port	The port which is the server listening on.
integer	
cors	CORS configuration for the HTTP Bridge.
`KafkaBridgeHttpCors`	

4.2.121. `KafkaBridgeHttpCors` schema reference

Used in: `KafkaBridgeHttpConfig`

Property	Description
allowedOrigins	List of allowed origins. Java regular expressions can be used.
string array	
allowedMethods	List of allowed HTTP methods.
string array	

4.2.122. `KafkaBridgeAdminClientSpec` schema reference

Used in: `KafkaBridgeSpec`

Property	Description
config	The Kafka AdminClient configuration used for AdminClient instances created by the bridge.
map	

4.2.123. `KafkaBridgeConsumerSpec` schema reference

Used in: `KafkaBridgeSpec`

Full list of `KafkaBridgeConsumerSpec` schema properties

Configures consumer options for the Kafka Bridge as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the Apache Kafka configuration documentation for consumers.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Kafka cluster bootstrap address
- Security (encryption, authentication, and authorization)
- Consumer group identifier

Properties with the following prefixes cannot be set:

- `bootstrap.servers`
- `group.id`
- `sasl.`
- `security.`
- `ssl.`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to Kafka Bridge, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites

Example Kafka Bridge consumer configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  consumer:
    config:
      auto.offset.reset: earliest
      enable.auto.commit: true
    # ...
```

IMPORTANT

The Cluster Operator does not validate keys or values in the `config` object. If an invalid configuration is provided, the Kafka Bridge deployment might not start or might become unstable. In this case, fix the configuration so that the Cluster Operator can roll out the new configuration to all Kafka Bridge nodes.

`KafkaBridgeConsumerSpec` schema properties

Property	Description
config	The Kafka consumer configuration used for consumer instances created by the bridge. Properties with the following prefixes cannot be set: ssl., bootstrap.servers, group.id, sasl., security. (with the exception of: ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols).
map	

4.2.124. `KafkaBridgeProducerSpec` schema reference

Used in: `KafkaBridgeSpec`

Full list of `KafkaBridgeProducerSpec` schema properties

Configures producer options for the Kafka Bridge as keys.

The values can be one of the following JSON types:

- String
- Number
- Boolean

Exceptions

You can specify and configure the options listed in the Apache Kafka configuration documentation for producers.

However, Strimzi takes care of configuring and managing options related to the following, which cannot be changed:

- Kafka cluster bootstrap address
- Security (encryption, authentication, and authorization)
- Consumer group identifier

Properties with the following prefixes cannot be set:

- `bootstrap.servers`
- `sasl.`
- `security.`
- `ssl.`

If the `config` property contains an option that cannot be changed, it is disregarded, and a warning message is logged to the Cluster Operator log file. All other supported options are forwarded to Kafka Bridge, including the following exceptions to the options configured by Strimzi:

- Any `ssl` configuration for supported TLS versions and cipher suites

Example Kafka Bridge producer configuration

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  producer:
    config:
      acks: 1
      delivery.timeout.ms: 300000
    # ...
```

IMPORTANT

The Cluster Operator does not validate keys or values in the `config` object. If an invalid configuration is provided, the Kafka Bridge deployment might not start or might become unstable. In this case, fix the configuration so that the Cluster Operator can roll out the new configuration to all Kafka Bridge nodes.

`KafkaBridgeProducerSpec` schema properties

Property	Description
config	The Kafka producer configuration used for producer instances created by the bridge. Properties with the following prefixes cannot be set: ssl., bootstrap.servers, sasl., security. (with the exception of: ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols).
map	

4.2.125. `KafkaBridgeTemplate` schema reference

Used in: `KafkaBridgeSpec`

Property	Description
deployment	Template for Kafka Bridge `Deployment`.
`DeploymentTemplate`	
pod	Template for Kafka Bridge `Pods`.
`PodTemplate`	
apiService	Template for Kafka Bridge API `Service`.
`InternalServiceTemplate`	

Property	Description
podDisruptionBudget PodDisruptionBudgetTemplate	Template for Kafka Bridge PodDisruptionBudget.
bridgeContainer ContainerTemplate	Template for the Kafka Bridge container.
clusterRoleBinding ResourceTemplate	Template for the Kafka Bridge ClusterRoleBinding.
serviceAccount ResourceTemplate	Template for the Kafka Bridge service account.
initContainer ContainerTemplate	Template for the Kafka Bridge init container.

4.2.126. KafkaBridgeStatus schema reference

Used in: KafkaBridge

Property	Description
conditions Condition array	List of status conditions.
observedGeneration integer	The generation of the CRD that was last reconciled by the operator.
url string	The URL at which external client applications can access the Kafka Bridge.
labelSelector string	Label selector for pods providing this resource.
replicas integer	The current number of pods being used to provide this resource.

4.2.127. KafkaConnector schema reference

Property	Description
spec KafkaConnectorSpec	The specification of the Kafka Connector.
status KafkaConnectorStatus	The status of the Kafka Connector.

4.2.128. KafkaConnectorSpec schema reference

Used in: KafkaConnector

Property	Description
class	The Class for the Kafka Connector.
string	
tasksMax	The maximum number of tasks for the Kafka Connector.
integer	
autoRestart	Automatic restart of connector and tasks configuration.
AutoRestart	
config	The Kafka Connector configuration. The following properties cannot be set: connector.class, tasks.max.
map	
pause	Whether the connector should be paused. Defaults to false.
boolean	

4.2.129. AutoRestart schema reference

Used in: KafkaConnectorSpec, KafkaMirrorMaker2ConnectorSpec

Full list of AutoRestart schema properties

Configures automatic restarts for connectors and tasks that are in a FAILED state.

When enabled, a back-off algorithm applies the automatic restart to each failed connector and its tasks.

The operator attempts an automatic restart on reconciliation. If the first attempt fails, the operator makes up to six more attempts. The duration between each restart attempt increases from 2 to 30 minutes. After each restart, failed connectors and tasks transit from FAILED to RESTARTING. If the restart fails after the final attempt, there is likely to be a problem with the connector configuration. The connector and tasks remain in a FAILED state and you have to restart them manually. You can do this by annotating the KafKaConnector custom resource with strimzi.io/restart: "true".

For Kafka Connect connectors, use the autoRestart property of the KafkaConnector resource to enable automatic restarts of failed connectors and tasks.

Enabling automatic restarts of failed connectors for Kafka Connect

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector
spec:
  autoRestart:
    enabled: true
```

For MirrorMaker 2, use the autoRestart property of connectors in the KafkaMirrorMaker2 resource to enable automatic restarts of failed connectors and tasks.

Enabling automatic restarts of failed connectors for MirrorMaker 2

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mm2-cluster
spec:
  mirrors:
  - sourceConnector:
      autoRestart:
        enabled: true
      # ...
    heartbeatConnector:
      autoRestart:
        enabled: true
      # ...
    checkpointConnector:
      autoRestart:
        enabled: true
      # ...
```

`AutoRestart` schema properties

Property	Description
enabled	Whether automatic restart for failed connectors and tasks should be enabled or disabled.
boolean	

4.2.130. `KafkaConnectorStatus` schema reference

Used in: `KafkaConnector`

Property	Description
conditions	List of status conditions.
`Condition` array	
observedGeneration	The generation of the CRD that was last reconciled by the operator.
integer	
autoRestart	The auto restart status.
`AutoRestartStatus`	
connectorStatus	The connector status, as reported by the Kafka Connect REST API.
map	
tasksMax	The maximum number of tasks for the Kafka Connector.
integer	

Property	Description
topics string array	The list of topics used by the Kafka Connector.

4.2.131. AutoRestartStatus schema reference

Used in: KafkaConnectorStatus, KafkaMirrorMaker2Status

Property	Description
count integer	The number of times the connector or task is restarted.
connectorName string	The name of the connector being restarted.
lastRestartTimestamp string	The last time the automatic restart was attempted. The required format is 'yyyy-MM-ddTHH:mm:ssZ' in the UTC time zone.

4.2.132. KafkaMirrorMaker2 schema reference

Property	Description
spec KafkaMirrorMaker2Spec	The specification of the Kafka MirrorMaker 2 cluster.
status KafkaMirrorMaker2Status	The status of the Kafka MirrorMaker 2 cluster.

4.2.133. KafkaMirrorMaker2Spec schema reference

Used in: KafkaMirrorMaker2

Property	Description
version string	The Kafka Connect version. Defaults to 3.4.0. Consult the user documentation to understand the process required to upgrade or downgrade the version.
replicas integer	The number of pods in the Kafka Connect group. Defaults to 3.
image string	The docker image for the pods.
connectCluster string	The cluster alias used for Kafka Connect. The alias must match a cluster in the list at spec.clusters.

Property	Description
clusters	Kafka clusters for mirroring.
`KafkaMirrorMaker2ClusterSpec` array	
mirrors	Configuration of the MirrorMaker 2 connectors.
`KafkaMirrorMaker2MirrorSpec` array	
resources	The maximum limits for CPU and memory resources and the requested initial resources. For more information, see the external documentation for core/v1 resourcerequirements.
ResourceRequirements	
livenessProbe	Pod liveness checking.
`Probe`	
readinessProbe	Pod readiness checking.
`Probe`	
jvmOptions	JVM Options for pods.
`JvmOptions`	
jmxOptions	JMX Options.
`KafkaJmxOptions`	
logging	Logging configuration for Kafka Connect. The type depends on the value of the `logging.type` property within the given object, which must be one of [inline, external].
`InlineLogging`, `ExternalLogging`	
clientRackInitImage	The image of the init container used for initializing the `client.rack`.
string	
rack	Configuration of the node label which will be used as the `client.rack` consumer configuration.
`Rack`	
tracing	The configuration of tracing in Kafka Connect. The type depends on the value of the `tracing.type` property within the given object, which must be one of [jaeger, opentelemetry].
`JaegerTracing`, `OpenTelemetryTracing`	
template	Template for Kafka Connect and Kafka Mirror Maker 2 resources. The template allows users to specify how the `Deployment`, `Pods` and `Service` are generated.
`KafkaConnectTemplate`	
externalConfiguration	Pass data from Secrets or ConfigMaps to the Kafka Connect pods and use them to configure connectors.
`ExternalConfiguration`	

Property	Description
metricsConfig	Metrics configuration. The type depends on the value of the `metricsConfig.type` property within the given object, which must be one of [jmxPrometheusExporter].
`JmxPrometheusExporterMetrics`	

4.2.134. `KafkaMirrorMaker2ClusterSpec` schema reference

Used in: `KafkaMirrorMaker2Spec`

Full list of `KafkaMirrorMaker2ClusterSpec` schema properties

Configures Kafka clusters for mirroring.

`config`

Use the `config` properties to configure Kafka options.

Standard Apache Kafka configuration may be provided, restricted to those properties not managed directly by Strimzi.

For client connection using a specific *cipher suite* for a TLS version, you can configure allowed `ssl` properties. You can also configure the `ssl.endpoint.identification.algorithm` property to enable or disable hostname verification.

`KafkaMirrorMaker2ClusterSpec` schema properties

Property	Description
alias	Alias used to reference the Kafka cluster.
string	
bootstrapServers	A comma-separated list of `host:port` pairs for establishing the connection to the Kafka cluster.
string	
tls	TLS configuration for connecting MirrorMaker 2 connectors to a cluster.
`ClientTls`	
authentication	Authentication configuration for connecting to the cluster. The type depends on the value of the `authentication.type` property within the given object, which must be one of [tls, scram-sha-256, scram-sha-512, plain, oauth].
`KafkaClientAuthenticationTls`, `KafkaClientAuthenticationScramSha256`, `KafkaClientAuthenticationScramSha512`, `KafkaClientAuthenticationPlain`, `KafkaClientAuthenticationOAuth`	

Property	Description
config	The MirrorMaker 2 cluster config. Properties with the following prefixes cannot be set: ssl., sasl., security., listeners, plugin.path, rest., bootstrap.servers, consumer.interceptor.classes, producer.interceptor.classes (with the exception of: ssl.endpoint.identification.algorithm, ssl.cipher.suites, ssl.protocol, ssl.enabled.protocols).
map	

4.2.135. KafkaMirrorMaker2MirrorSpec schema reference

Used in: KafkaMirrorMaker2Spec

Property	Description
sourceCluster	The alias of the source cluster used by the Kafka MirrorMaker 2 connectors. The alias must match a cluster in the list at spec.clusters.
string	
targetCluster	The alias of the target cluster used by the Kafka MirrorMaker 2 connectors. The alias must match a cluster in the list at spec.clusters.
string	
sourceConnector	The specification of the Kafka MirrorMaker 2 source connector.
KafkaMirrorMaker2ConnectorSpec	
heartbeatConnector	The specification of the Kafka MirrorMaker 2 heartbeat connector.
KafkaMirrorMaker2ConnectorSpec	
checkpointConnector	The specification of the Kafka MirrorMaker 2 checkpoint connector.
KafkaMirrorMaker2ConnectorSpec	
topicsPattern	A regular expression matching the topics to be mirrored, for example, "topic1|topic2|topic3". Comma-separated lists are also supported.
string	
topicsBlacklistPattern	**The topicsBlacklistPattern property has been deprecated, and should now be configured using .spec.mirrors.topicsExcludePattern.** A regular expression matching the topics to exclude from mirroring. Comma-separated lists are also supported.
string	
topicsExcludePattern	A regular expression matching the topics to exclude from mirroring. Comma-separated lists are also supported.
string	
groupsPattern	A regular expression matching the consumer groups to be mirrored. Comma-separated lists are also supported.
string	

Property	Description
groupsBlacklistPattern	**The `groupsBlacklistPattern` property has been deprecated, and should now be configured using `.spec.mirrors.groupsExcludePattern`.** A regular expression matching the consumer groups to exclude from mirroring. Comma-separated lists are also supported.
string	
groupsExcludePattern	A regular expression matching the consumer groups to exclude from mirroring. Comma-separated lists are also supported.
string	

4.2.136. `KafkaMirrorMaker2ConnectorSpec` schema reference

Used in: `KafkaMirrorMaker2MirrorSpec`

Property	Description
tasksMax	The maximum number of tasks for the Kafka Connector.
integer	
config	The Kafka Connector configuration. The following properties cannot be set: connector.class, tasks.max.
map	
autoRestart	Automatic restart of connector and tasks configuration.
`AutoRestart`	
pause	Whether the connector should be paused. Defaults to false.
boolean	

4.2.137. `KafkaMirrorMaker2Status` schema reference

Used in: `KafkaMirrorMaker2`

Property	Description
conditions	List of status conditions.
`Condition` array	
observedGeneration	The generation of the CRD that was last reconciled by the operator.
integer	
url	The URL of the REST API endpoint for managing and monitoring Kafka Connect connectors.
string	
autoRestartStatuses	List of MirrorMaker 2 connector auto restart statuses.
`AutoRestartStatus` array	

Property	Description
connectorPlugins `ConnectorPlugin` array	The list of connector plugins available in this Kafka Connect deployment.
connectors map array	List of MirrorMaker 2 connector statuses, as reported by the Kafka Connect REST API.
labelSelector string	Label selector for pods providing this resource.
replicas integer	The current number of pods being used to provide this resource.

4.2.138. `KafkaRebalance` schema reference

Property	Description
spec `KafkaRebalanceSpec`	The specification of the Kafka rebalance.
status `KafkaRebalanceStatus`	The status of the Kafka rebalance.

4.2.139. `KafkaRebalanceSpec` schema reference

Used in: `KafkaRebalance`

Property	Description
mode string (one of [remove-brokers, full, add-brokers])	Mode to run the rebalancing. The supported modes are `full`, `add-brokers`, `remove-brokers`. If not specified, the `full` mode is used by default. • `full` mode runs the rebalancing across all the brokers in the cluster. • `add-brokers` mode can be used after scaling up the cluster to move some replicas to the newly added brokers. • `remove-brokers` mode can be used before scaling down the cluster to move replicas out of the brokers to be removed.
brokers integer array	The list of newly added brokers in case of scaling up or the ones to be removed in case of scaling down to use for rebalancing. This list can be used only with rebalancing mode `add-brokers` and `removed-brokers`. It is ignored with `full` mode.

Property	Description
goals	A list of goals, ordered by decreasing priority, to use for generating and executing the rebalance proposal. The supported goals are available at https://github.com/linkedin/cruise-control#goals. If an empty goals list is provided, the goals declared in the default.goals Cruise Control configuration parameter are used.
string array	
skipHardGoalCheck	Whether to allow the hard goals specified in the Kafka CR to be skipped in optimization proposal generation. This can be useful when some of those hard goals are preventing a balance solution being found. Default is false.
boolean	
rebalanceDisk	Enables intra-broker disk balancing, which balances disk space utilization between disks on the same broker. Only applies to Kafka deployments that use JBOD storage with multiple disks. When enabled, inter-broker balancing is disabled. Default is false.
boolean	
excludedTopics	A regular expression where any matching topics will be excluded from the calculation of optimization proposals. This expression will be parsed by the java.util.regex.Pattern class; for more information on the supported format consult the documentation for that class.
string	
concurrentPartitionMovementsPerBroker	The upper bound of ongoing partition replica movements going into/out of each broker. Default is 5.
integer	
concurrentIntraBrokerPartitionMovements	The upper bound of ongoing partition replica movements between disks within each broker. Default is 2.
integer	
concurrentLeaderMovements	The upper bound of ongoing partition leadership movements. Default is 1000.
integer	
replicationThrottle	The upper bound, in bytes per second, on the bandwidth used to move replicas. There is no limit by default.
integer	
replicaMovementStrategies	A list of strategy class names used to determine the execution order for the replica movements in the generated optimization proposal. By default BaseReplicaMovementStrategy is used, which will execute the replica movements in the order that they were generated.
string array	

4.2.140. KafkaRebalanceStatus schema reference

Used in: KafkaRebalance

Property	Description
conditions	List of status conditions.
Condition array	
observedGeneration	The generation of the CRD that was last reconciled by the operator.
integer	
sessionId	The session identifier for requests to Cruise Control pertaining to this KafkaRebalance resource. This is used by the Kafka Rebalance operator to track the status of ongoing rebalancing operations.
string	
optimizationResult	A JSON object describing the optimization result.
map	