Cloudera Streams Messaging Operator for Kubernetes 1.5.0

Cloudera Surveyor for Apache Kafka Configuration

Date published: 2024-06-11
Date modified: 2025-10-30

https://docs.cloudera.com/

Legal Notice

Contents

Cloudera Surveyor for Apache Kafka configuration overview

Get started with configuring Cloudera Surveyor. Learn about basic configuration practices using Helm and available configuration properties.

Cloudera Surveyor is exclusively managed using Helm. You initially configure Cloudera Surveyor during installation. Typically this involves creating a custom values file (values.yml) that includes configuration properties. The file is applied during installation when you run the helm install command.

If required, you can make configuration updates following installation. This is done with the helm upgrade command using the --reuse-values option together with the -f (--values) or --set options.

```
helm upgrade CLOUDERA-SURVEYOR [***CHART***] \
  --namespace [***NAMESPACE***] \
  (--set '[***KEY***]=[***VALUE***]' | -f [***MY-VALUES.YAML***] | --set-fil
e [***KEY***]=[***FILEPATH***]) \
  --reuse-values
```

- The string cloudera-surveyor is the Helm release name of the chart installation. This is an arbitrary, user defined name.
- Ensure that *[***CHART***]* and *[***NAMESPACE***]* are the same as the ones you used during installation. You can use helm list to list currently installed releases and charts.
- Use --set if you want to update properties directly from the command line. Helm supports various --set options like --set-file, --set-string, and others. You can use any of these options.
- Use -f (--values) if you want to pass a file containing your configuration updates.
- The --reuse-values option instructs Helm to merge existing values with new ones. You use this option when you want to update an existing configuration.

Configurable properties

Cloudera Surveyor accepts various configuration properties. You can find a comprehensive list of these properties in the *Helm chart configuration reference*. Alternatively, you can list available properties with helm show readme.

```
helm show readme [***CHART***]
```

Related Information

Cloudera Surveyor Helm chart configuration reference
Helm List | Helm
Helm Upgrade | Helm
Helm Show Readme | Helm

Registering Kafka clusters in Cloudera Surveyor for Apache Kafka

Learn how to register Kafka clusters. Registering a Kafka cluster enables management and monitoring of the cluster through Cloudera Surveyor. You can register any Kafka cluster that is compatible with the Apache Kafka 2.4.1 API or higher.

You register a Kafka cluster with clusterConfigs.* properties. These properties specify the Kafka clusters that Cloudera Surveyor connects to. After a cluster is registered, you are able to manage and monitor the cluster through

Cloudera Surveyor. This includes the ability to view cluster information and execute supported management tasks, such as viewing cluster status and health, as well as managing topics and consumer groups.

Cloudera Surveyor connects to Kafka clusters as any other Kafka client. Therefore, the configuration you specify with clusterConfigs.* properties will be similar to standard Kafka client configuration.

Specifically clusterConfigs.clusters is an array of connected clusters. Each item in this array is a map that defines the configuration for a single Kafka cluster, including properties such as clusterName, bootstrapServers, tags, and comm onClientConfig. Cloudera Surveyor can connect to any Kafka cluster that provides an API compatible with Apache Kafka 2.4.1 or higher.

The following is a simple clusterConfigs.* example that registers an unsecured Kafka cluster with some tags configured.

```
#...
clusterConfigs:
  clusters:
    - clusterName: "[***CLUSTER NAME***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      tags:
        - "[***TAG1***]"
        - "[***TAG2***]"
      commonClientConfig:
        security.protocol: PLAINTEXT
```

- clusterConfigs.clusters[*] – An array of Kafka clusters and their configuration. Each entry defines the configuration for a single Kafka cluster.
- clusterConfigs.clusters[*].clustername – The name of the cluster. This name is displayed on the UI.
- clusterConfigs.clusters[*].bootstrapServers – A comma-separated list of the bootstrap servers for the Kafka cluster that Cloudera Surveyor connects to. Specify multiple servers for highly available connections.
- clusterConfigs.clusters[*].tags – User defined tags. Used for organization and filtering.
- clusterConfigs.cluster[*].commonClientConfig – Kafka client configuration properties applied to all clients for this cluster. Must contain upstream Kafka client properties as a map.

In addition to standard upstream Kafka client properties, clusterConfigs.* also accepts various properties that are specific to Cloudera Surveyor. These properties allow you to configure tags, snapshot intervals, alert thresholds, and more on a per-cluster basis. For a full list, see *Cloudera Surveyor Helm chart configuration reference*.

Note: When using helm upgrade, clusterConfigs properties are overridden and not merged, even if the --reuse-values option is specified. Always provide the full configuration for clusterConfigs during updates. Otherwise, previously set configuration might be lost. As a best practice, save a values file containing your complete clusterConfigs configuration. If you need to make changes, update this file and specify it with --va lues when running helm upgrade. Additionally use the --reuse-values option to retain other properties.

Related Information

Cloudera Surveyor Helm chart configuration reference

Registering secure clusters

When registering a Kafka cluster that has security enabled, you must provide security-related client properties in your configuration. The exact Kafka client properties you specify depend on the security configuration of the Kafka cluster you want to register.

The following example snippets demonstrate various clusterConfigs.* examples for Kafka clusters that have some of the most commonly used security setups.

In these examples, sensitive data is mounted to the filesystem from Kubernetes `Secrets`. Sensitive data is then specified in the configuration using references. The references are resolved by Cloudera Surveyor with the Kafka `DirectoryConfigProvider`. This way, sensitive data is not stored in the configuration file, but rather in a

secure location that can be referenced at runtime. For more information on handling sensitive data in configurations, see Managing sensitive data in client configuration on page 8.

For PLAIN with TLS

```
#...
clusterConfigs:
  clusters:
    - clusterName: "[***CLUSTER NAME***]"
      tags:
        - "[***TAG1***]"
        - "[***TAG2***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      commonClientConfig:
        security.protocol: "SASL_SSL"
        sasl.mechanism: PLAIN
        ssl.truststore.type: "pkcs12"
        ssl.truststore.location: "/opt/secrets/[***TRUSTSTORE
 SECRET***]/[***TRUSTSTORE FILE***]"
        ssl.truststore.password: "\\${dir:/opt/secrets/[***TRUSTSTORE
 SECRET***]:[***TRUSTSTORE PASSWORD FILE***]}"
        sasl.jaas.config: "\\${dir:/opt/secrets/[***JAAS.CONF
 SECRET***]:[***JAAS.CONF***]}"
secretsToMount:
  - create: false
    secretRef: "[***TRUSTSTORE SECRET***]"
    items:
      - key: "[***TRUSTSTORE PASSWORD KEY***]"
        path: "[***TRUSTSTORE PASSWORD FILE***]"
      - key: "[***TRUSTSTORE KEY***]"
        path: "[***TRUSTSTORE FILE***]"
  - create: false
    secretRef: "[***JAAS.CONF SECRET***]"
    items:
      - key: "[***JAAS.CONF KEY***]"
        path: "[***JAAS.CONF***]"
```

For Kerberos with TLS

```
#...
clusterConfigs:
  clusters:
    - clusterName: "[***CLUSTER NAME***]"
      tags:
        - "[***TAG1***]"
        - "[***TAG2***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      commonClientConfig:
        security.protocol: "SASL_SSL"
        sasl.mechanism: GSSAPI
        sasl.kerberos.service.name: [***KAFKA SERVICE NAME***]
        ssl.truststore.type: "pkcs12"
        ssl.truststore.location: "/opt/secrets/[***TRUSTSTORE
 SECRET***]/[***TRUSTSTORE FILE***]"
        ssl.truststore.password: "\\${dir:/opt/secrets/[***TRUSTSTORE
 SECRET***]:[***TRUSTSTORE PASSWORD FILE***]}"
        sasl.jaas.config: "\\${dir:/opt/secrets/[***JAAS & KEYTAB
 SECRET***]:[***JAAS.CONF***]}"
secretsToMount:
  - create: false
    secretRef: "[***TRUSTSTORE SECRET***]"
    items:
```

```
      - key: "[***TRUSTSTORE PASSWORD KEY***]"
        path: "[***TRUSTSTORE PASSWORD FILE***]"
      - key: "[***TRUSTSTORE KEY***]"
        path: "[***TRUSTSTORE FILE***]"
  - create: false
    secretRef: "[***JAAS & KEYTAB SECRET***]"
    items:
      - key: "[***JAAS.CONF KEY***]" # the keytab in the jaas.conf must
 point to /opt/secrets/[***JAAS & KEYTAB SECRET***]/[***KAFKA.KEYTAB***]
        path: "[***JAAS.CONF***]"
      - key: "[***KAFKA.KEYTAB KEY***]"
        path: "[***KAFKA.KEYTAB***]"
```

For OAUTH2 with TLS

```
#...
clusterConfigs:
  clusters:
    - clusterName: "[***CLUSTER NAME***]"
      tags:
        - "[***TAG1***]"
        - "[***TAG2***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      commonClientConfig:
        security.protocol: "SASL_SSL"
        sasl.mechanism: OAUTHBEARER
        sasl.login.callback.handler.class: "org.apache.kafka.common.securi
ty.oauthbearer.secured.OAuthBearerLoginCallbackHandler"
        sasl.oauthbearer.token.endpoint.url: "[***OAUTH TOKEN ENDPOINT
 URL***]"
        ssl.truststore.type: "pkcs12"
        ssl.truststore.location: "/opt/secrets/[***TRUSTSTORE
 SECRET***]/[***TRUSTSTORE FILE***]"
        ssl.truststore.password: "\\${dir:/opt/secrets/[***TRUSTSTORE
 SECRET***]:[***TRUSTSTORE PASSWORD FILE***]}"
        sasl.jaas.config: "\\${dir:/opt/secrets/[***JAAS.CONF
 SECRET***]:[***JAAS.CONF***]}"
secretsToMount:
  - create: false
    secretRef: "[***TRUSTSTORE SECRET***]"
    items:
      - key: "[***TRUSTSTORE PASSWORD SECRET***]"
        path: "[***TRUSTSTORE PASSWORD FILE***]"
      - key: "[***TRUSTSTORE KEY***]"
        path: "[***TRUSTSTORE FILE***]"
  - create: false
    secretRef: "[***JAAS.CONF SECRET***]"
    items:
      - key: "[***JAAS.CONF KEY***]"
        path: "[***JAAS.CONF***]"
```

Bootstrap servers for Kafka deployed in Kubernetes

If Cloudera Surveyor for Apache Kafka and the Kafka cluster you want to register are deployed in the same Kubernetes cluster, you can use the DNS name of the Kubernetes `Service` that provides access to the Kafka cluster as the bootstrap server.

For example, the DNS name of the default `ClusterIP Service` for a Kafka cluster that was deployed with the Strimzi Cluster Operator is similar to the following example.

```
my-cluster-kafka-bootstrap.my-namespace.svc.cluster.local
```

Where my-cluster is the name of the Kafka cluster, kafka-bootstrap is a fixed affix, my-namespace is the namespace of the cluster, and svc.cluster.local is the domain name used internally by the Kubernetes cluster.

Managing sensitive data in client configuration

Learn about storing, managing, and referencing sensitive data in the Kafka client properties you configure for Cloudera Surveyor.

When you register a secure Kafka cluster, you must provide Cloudera Surveyor with Kafka client properties that make connection to the cluster possible. If the Kafka cluster is secure, the properties that you specify include sensitive data such as credentials, authentication tokens, certificates, and others.

Instead of hard-coding sensitive data in your configuration, Cloudera Surveyor supports mounting data from `Secrets` that are in the same namespace. Data mounted this way can be referenced in your configuration. References are resolved with the Kafka `DirectoryConfigProvider`.

Mounting Secrets

Use the secretsToMount property to specify which `Secrets` and keys from a `Secret` you want to mount. The `Secret` must be in the same namespace where Cloudera Surveyor is installed. The following example mounts a single key from a single `Secret`.

```
#...
secretsToMount:
  - create: false
    secretRef: "[***SECRET NAME***]"
    items:
      - key: "[***SECRET KEY***]"
        path: "[***PATH TO MOUNT DATA***]"
```

Each item inside secretsToMount must have a secretRef property, which specifies the name of the `Secret` to mount. Optionally, you can include an items array, which maps `Secret` keys to paths. If not present, all keys from the `Secret` are mounted as is.

Data is mounted to /opt/secrets/*[***SECRET NAME***]*/ in the Cloudera Surveyor `Container`. This means that the value you specify in secretsToMount[*].items[*].path is relative to /opt/secrets/*[***SECRET NAME***]*/.

Referencing mounted data

All Kafka clients used by Cloudera Surveyor use the Kafka `DirectoryConfigProvider`. Use the following syntax to reference mounted data as the value for a client property.

```
${dir:[***FULL DIR PATH***]:[***FILENAME***]}
```

For example, assume you have a `Secret` named my-kafka-jaas-secret. This `Secret` contains a single key named jaas-key. The key contains a JAAS configuration. To mount the `Secret` and reference its data, you pass the following configuration.

```
#...
clusterConfigs:
  clusters:
    - clusterName: "my-kafka"
      commonClientConfig:
```

```
        sasl.jaas.config: "\\${dir:/opt/secrets/my-kafka-jaas-secret/jaas.c
onf}"
secretsToMount:
  - create: false
    secretRef: "my-kafka-jaas-secret"
    items:
      - key: "jaas-key"
        path: "jaas.conf"
```

In this example, the value of the jaas-key key in the `Secret` is mounted to /opt/secrets/my-kafka-jaas-secret/jaas.conf. This file is referenced as the value for the sasl.jaas.config Kafka client property using `DirectoryConfigProvider` syntax. As a result, the Kafka client in Cloudera Surveyor that connects to the my-kafka cluster uses the specified JAAS configuration for authentication.

Note: References in the client configurations must be escaped because Cloudera Surveyor itself uses the same syntax for references.

Creating Secrets

If a `Secret` that you want to mount does not exist, you can create it by setting the secretsToMount[*].create property to true. In this case, the specified `Secret` is created and managed by Helm. The content for each item in the items array is set through the secretsToMount[*].items[*].content property.

Because the content property must include your actual data in plaintext, Cloudera recommends that you do not set this property directly in your custom values file (values.yaml). Instead, pass the contents of your `Secrets` with the --set-file option when you run a helm install or helm upgrade command. This allows you to reference the contents of the `Secrets` from a file. This way, sensitive data is not made available in shell history or stored directly in your values file.

For example, assume you specify the following in your values file.

```
#...
secretsToMount:
  - create: true
    secretRef: "my-kafka-jaas-secret"
    items:
      - key: "jaas-key"
        path: "jaas.conf"
```

Notice that content is not specified. When applying this configuration with helm, you pass the contents of jaas-key in my-kafka-jaas-secret using --set-file.

```
helm upgrade CLOUDERA-SURVEYOR [***CHART***] \
  --namespace [***NAMESPACE***] \
  --values [***MY-VALUES.YAML***] \
  --set-file secretsToMount[0].items[0].content=[***FILEPATH***] \
  --reuse-values
```

- Ensure that you use correct numeric indices ([0]) to target specific list items. secretsToMount can have multiple `Secrets`, and each `Secret` can have multiple keys.
- *[***FILEPATH***]* in this case points to a file containing the JAAS configuration.

Client pools and client configuration hierarchy

Learn about the Kafka client pools used by Cloudera Surveyor and the configuration hierarchy that determines how client properties are applied.

Cloudera Surveyor uses two separate pools of Kafka clients to interact with Kafka clusters. The pools are as follows.

- **Snapshot pool** – Used for periodic read operations. These clients collect cluster state, topic metadata, and consumer group information from clusters.
- **Admin pool** – Used for administrative tasks, such as creating, deleting, or updating topics and consumer groups.

You can configure each pool of clients separately and on multiple configuration levels using various properties.

Configuration levels and hierarchy

Kafka client configuration is applied in a hierarchical order, from the least specific to most specific. Configurations are also merged. If duplicate keys exist, the value from the more specific configuration overrides the value from the less specific one.

This configuration hierarchy provides granular control over each client that Cloudera Surveyor uses. In addition, it enables you to specify common properties at higher levels, reducing redundancy and simplifying management in configuration.

The order from least to most specific levels is as follows.

- **Global common** – Client configuration applied to all Kafka clients that Cloudera Surveyor uses. Configured with surveyorConfig.surveyor.commonClientConfig.
- **Global pool** – Client configuration applied to all Kafka clients belonging to the specified pool. Configured with surveyorConfig.surveyor.snapshotClientPool.clientConfig and surveyorConfig.surveyor.adminClientPool.clientConfig.
- **Cluster common** – Client configuration applied to Kafka clients used for the specified cluster. Configured with clusterConfigs.clusters[*].commonClientConfig.
- **Cluster pool** – Client configuration applied to all Kafka clients belonging to the specified pool that connect to the specified cluster. Configured with clusterConfigs.clusters[*].snapshotClientPool.clientConfig and clusterConfigs.clusters[*].adminClientPool.clientConfig

For example, assume you pass the following configuration to Cloudera Surveyor.

```
#...
surveyorConfig:
  surveyor:
    commonClientConfig:
      security.protocol: SASL_SSL
      sasl.mechanism: PLAIN
  snapshotClientPool:
    clientConfig:
      request.timeout.ms: 30000
  adminClientPool:
    clientConfig:
      retries: 5

clusterConfigs:
  clusters:
    - clusterName: "[***CLUSTER NAME***]"
      tags:
        - "[***TAG1***]"
        - "[***TAG2***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      commonClientConfig:
        security.protocol: "PLAINTEXT"
      snapshotClientPool:
        clientConfig:
          request.timeout.ms: 10000
```

The resulting client configuration used by the clients is as follows.

- For the snapshot clients connecting to the *[***CLUSTER NAME***]* cluster:
  - security.protocol is PLAINTEXT (cluster common override).
  - request.timeout.ms is 10000 (cluster pool override).
- For the admin clients connecting to the *[***CLUSTER NAME***]* cluster:
  - security.protocol is PLAINTEXT (cluster common override).
  - retries is 5 (no override).

Configuring external access in Cloudera Surveyor for Apache Kafka

Learn how you can configure Cloudera Surveyor to provide secure external access to its UI.

Cloudera Surveyor provides a web-based UI that users access externally. By default the UI is exposed using a `NodePort` type Kubernetes `Service` that is unsecured.

To further configure and secure external access, you can configure a Kubernetes `Ingress` on top of the `NodePort`. Alternatively, you can deploy a `LoadBalancer` type `Service` instead of the `Nodeport`. Both methods allow you to provide external users with secure (TLS) access to the UI. The choice between `Ingress` and `LoadBalancer` depends on your infrastructure, security requirements, and need for advanced routing or certificate management.

Note: While both methods allow for both encrypted (TLS) and unencrypted communication. Cloudera recommends that you always enable encrypted communications with TLS for external access.

Configuring external access with Ingress

Learn how to configure external access to the Cloudera Surveyor UI with a Kubernetes Ingress.

Before you begin

- An `Ingress` controller is required. Ensure that you have one deployed in your Kubernetes cluster. For example, you can use the Ingress-Nginx controller.
- Optional: cert-manager is installed in your Kubernetes cluster.

  Although not required, cert-manager enables you to manage certificates automatically. Without cert-manager you must manage your certificate manually through `Secrets`. The following steps assume that cert-manager is available.

Procedure

1. Deploy an `Issuer` resource for cert-manager.

   Take note of the name of the `Issuer` you deploy. You provide the name of the `Issuer` to the `Ingress` in a following step. Deploying a `Certificate` resource is not needed, it is automatically requested and created by the `Ingress` once it is deployed.
2. Configure ingress properties in a valiues file (values.yaml).

```
#...
ingress:
  enabled: true
  className: "nginx"
  rules:
    host: MY-APP.EXAMPLE.CLOUDERA.COM
    port: 443
  tls:
```

```
    enabled: true
    secretRef: "[***INGRESS TLS CERT SECRET***]"
  extraAnnotations:
    cert-manager.io/issuer: "[***ISSUER NAME***]"
```

- ingress.enabled – Enables or disables `Ingress`.
- ingress.className – The class name of the `Ingress` controller. This example configures the Ingress-Nginx controller.
- ingress.rules.host – Specifies the DNS hostname that the `Ingress` controller should match for incoming HTTP/HTTPS requests.
- ingress.rules.port – The port of the `Ingress` rule. This is the port of the Kubernetes `Service` that the `Ingress` forwards requests to.
- ingress.tls.enabled – Enables TLS for the `Ingress`.
- ingress.tls.secretRef –The name of the `Secret` that contains the Ingress TLS certificates. When using cert-manager and the cert-manager.io/issuer annotation is set in the ingress.extraAnnotations property, a certificate is requested automatically and saved to the `Secret` specified here.
- ingress.extraAnnotations.* – Extra annotations to apply to the `Ingress`.

  The cert-manager.io/issuer annotation configures the name of the cert-manager `Issuer`. When set, a certificate is automatically requested by the `Ingress`.

3. Apply configuration changes.

```
helm upgrade CLOUDERA-SURVEYOR [***CHART***] \
  --namespace [***NAMESPACE***] \
  --values [***VALUES.YAML***] \
  --reuse-values
```

4. Access the UI.

   The UI is accessible from the Hostname/IP of the `Ingress`.

```
kubectl get ingress --namespace [***NAMESPACE***]
```

```
NAME                CLASS   HOSTS                  ADDRESS     PORTS
#...
cloudera-surveyor-ingress   nginx   my-app.example.cloudera.com  10.14.
91.1  80, 443
```

   In this example, the UI will be accessible on my-app.cloudera.com:443.

Configuring external access with LoadBalancer

Learn how to configure external access to the Cloudera Surveyor UI with a LoadBalancer type Service.

Before you begin

When deploying a `LoadBalancer` type `Service`, the actual load balancer is provisioned and managed by your cloud or infrastructure provider. As a result, TLS settings and certificate management may vary depending on the platform. Refer to vendor-specific documentation for detailed guidance on configuring TLS.

Procedure

1. Set service.type to LoadBalancer in a custom values file (values.yaml).

```
#...
service:
  type: LoadBalancer
```

```
  port: 8080
  targetPort: 8080
  tlsPort: 8443
  tlsTargetPort: 8443
```

Note: Ensure that `Ingress` is disabled (ingress.enabled: false). `Ingress` is disabled by default.

2. Apply configuration changes.

```
helm upgrade CLOUDERA-SURVEYOR [***CHART***] \
  --namespace [***NAMESPACE***] \
  --values [***VALUES.YAML***] \
  --reuse-values
```

3. Access the UI.

   The UI is accessible from the Hostname/IP of the load balancer.

```
kubectl get service SURVEYOR-service --namespace [***NAMESPACE***]
```

   Look at the IP listed in the EXTERNAL-IP column as well as the port in the PORT(S) column. You can access the UI through this IP and port.

```
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP        PORT
(S)
cloudera-surveyor-service  LoadBalancer   10.103.58.116   104.198.205.71
   8080:30219/TCP
```

   In this example, the UI will be accessible on 104.198.205.71:30219.