Cloudera Streams Messaging Operator for Kubernetes 1.7.0

Cloudera Surveyor for Apache Kafka Configuration

Date published: 2024-06-11
Date modified: 2026-07-31

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).

Displayed broker addresses

The broker addresses you see in the Cloudera Surveyor UI come from the Kafka listener that Cloudera Surveyor's clusterConfigs.clusters[*].bootstrapServers property points to. The clusterConfigs.clusters[*].brokerAddressListener property overrides this to display addresses from a different listener.

Cloudera Surveyor connects to each registered Kafka cluster using its Kafka clients. These clients connect to the cluster through the address specified in clusterConfigs.clusters[*].bootstrapServers. During each snapshot, Cloudera Surveyor calls the Kafka describeCluster API to collect broker metadata, including the address of each broker in the cluster.

Kafka returns broker addresses (host and port) that correspond to the listener that the clients connected through, that is, the listener that the clusterConfigs.clusters[*].bootstrapServers address belongs to. The returned host and port pairs are displayed in the **Address** attribute on **Broker Details** pages. Additionally, the host and port values are visible individually in the **Hostname** and **Port** attributes on **Broker** pages in the **Entity Details** drawer of individual brokers.

In many deployments, the clusterConfigs.clusters[*].bootstrapServers address and the broker addresses you want displayed in the UI belong to the same listener. In this case, no additional configuration is needed.

However, Kafka brokers can be configured with multiple named listeners, each with a different address. In environments where this is the case, the listener used for the bootstrap connection may not expose the broker addresses you want displayed. For example, Cloudera Surveyor might connect to Kafka through an internal listener, while users need to see a different listener's addresses in the UI to connect their own applications.

In these cases, you can use the clusterConfigs.clusters[*].brokerAddressListener property to specify which listener's addresses Cloudera Surveyor should display.

When configured, Cloudera Surveyor looks up the specified listener name in each broker's advertised.listeners configuration, falling back to listeners if advertised.listeners is not set. If the specified listener name is not found in either configuration, Cloudera Surveyor falls back to the address returned by describeCluster for that broker.

Customizing displayed broker addresses

Customize what broker addresses are displayed in Cloudera Surveyor with clusterConfigs.clusters[*].brokerAddress Listener.

Set clusterConfigs.clusters[*].brokerAddressListener to the name of the Kafka listener whose broker addresses you want Cloudera Surveyor to display. The name of a Kafka listener corresponds to the label before :// in the broker's advertised.listeners configuration. For example, in the following advertised.listeners value, the available listener names are PLAIN-9092 and EXTERNAL-9094.

```
advertised.listeners=PLAIN-9092://broker1.internal:9092,EXTERNAL-9094://brok
er1.example.com:8443
```

Note: In Kafka clusters managed by the Strimzi Cluster Operator, listener names in advertised.listeners follow the format *[***LISTENER NAME***]-[***PORT***]*, where the listener name and port are combined with a hyphen. This is a Strimzi convention and might not apply to clusters deployed with other systems or tooling.

The following example configures Cloudera Surveyor to display the EXTERNAL-9094 listener addresses from the example above.

```
#...
clusterConfigs:
  clusters:
```

```
    - clusterName: "[***CLUSTER NAME***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      brokerAddressListener: "EXTERNAL-9094"
      commonClientConfig:
        security.protocol: PLAINTEXT
```

Disabling broker configuration editing

Disable broker configuration editing for a registered Kafka cluster by setting the readOnlyBrokerConfigs property to true in your values file.

The clusterConfigs.clusters[*].readOnlyBrokerConfigs property controls whether you can modify broker configuration properties for a registered Kafka cluster through the Cloudera Surveyor UI or API. When set to true, the Edit Properties button on **Broker Details** pages is disabled and the API rejects any attempt to modify broker configurations. The property defaults to false, which allows editing.

This property is particularly useful for Kafka clusters managed by the Strimzi Cluster Operator. Strimzi continuously reconciles broker configurations from the `Kafka` custom resource. Any changes you apply through Cloudera Surveyor are overwritten during the next reconciliation cycle. Cloudera recommends setting readOnlyBrokerConfigs to true for Strimzi-managed clusters. More broadly, Cloudera recommends using this property for any Kafka cluster whose configuration is owned by an external lifecycle management tool, automation, or provisioning system.

Set readOnlyBrokerConfigs: true on the cluster you want to make read-only:

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

Exposing client configurations for download

Cloudera Surveyor exposes a per-cluster bundle of files on the Client Configurations tab of the Cluster Details page. End users can preview, copy, and download the files they need to connect external clients to a registered Kafka cluster.

End users of Cloudera Surveyor might need an easy way to download the files required to connect an external client to a specific Kafka cluster. You can configure Cloudera Surveyor to expose a bundle of such files on the Cluster Details Client Configurations tab. From the tab, end users can preview file content, copy it to the clipboard, and download individual files or the full archive.

The primary use case is making Kafka client connection material easily accessible in one place, such as properties files, CA certificates, and usage instructions, without having to distribute them manually. You control what appears on the tab: the choice of files, their display names, and their descriptions in the UI. You can also surface any other cluster-related file, not only Kafka client configurations.

You configure this with the clusterConfigs.clusters[*].clientConfigs properties in your values file. The clientCo nfigs block is optional and per-cluster. Add it only to the clusters you want to expose files for. When clientConfigs is omitted for a cluster, the **Client Configurations** tab is empty for that cluster.

You store file content in Kubernetes `Secrets` in the same namespace as the Cloudera Surveyor release. You reference each file with a secretRef in clientConfigs.files, specifying the `Secret` name and the key within the `Secret` that holds the file content. The Cloudera Surveyor Helm chart automatically generates the pod volume and

mount for each referenced `Secret`. No manual extraVolumes or extraVolumeMounts entries are required for client configuration files.

Important:

clientConfigs is not related to commonClientConfig or secretsToMount. Those properties configure how Cloudera Surveyor itself connects to Kafka. clientConfigs is solely for files that you expose to end users on the **Client Configurations** tab.

Related Information

Cloudera Surveyor Clusters

Creating Kubernetes Secrets for client configuration files

Create the Kubernetes Secrets that hold the file content for the client configuration bundle you want to expose in Cloudera Surveyor.

You store each file's content as a key in a Kubernetes `Secret`. You can group multiple files into a single `Secret` or use one `Secret` per logical source, such as one for client configuration properties files and one for TLS certificates. The key name in the `Secret` must match the secretRef.key value you declare in your values file.

`Secrets` must be in the same namespace as the Cloudera Surveyor release. If a `Secret` is already managed by cert-manager or a Kafka operator, you can reference it directly in your values file without creating a new one. When the `Secret` name and key remain the same, certificate rotations propagate to Cloudera Surveyor without a Helm upgrade.

Important: Do not add extraVolumes or extraVolumeMounts entries for client configuration files. The Cloudera Surveyor Helm chart automatically generates the pod volume and mount from the secretRef entries you declare in clientConfigs.

Create a `Secret` for each file or group of files you want to include in the bundle. The following examples use kube ctl to create two `Secrets`: one holding a client configuration properties file and a README, and one holding a CA certificate.

```
kubectl create secret generic [***CLIENT CONFIG SECRET NAME***] \
  --namespace [***NAMESPACE***] \
  --from-file=external.properties=[***PATH TO EXTERNAL.PROPERTIES***] \
  --from-file=README.md=[***PATH TO README.MD***]

kubectl create secret generic [***TLS SECRET NAME***] \
  --namespace [***NAMESPACE***] \
  --from-file=ca.crt=[***PATH TO CA CERTIFICATE***]
```

Alternatively, use a YAML manifest to create the same `Secrets`:

```
apiVersion: v1
kind: Secret
metadata:
  name: [***CLIENT CONFIG SECRET NAME***]
  namespace: [***NAMESPACE***]
type: Opaque
stringData:
  external.properties: |
    [***CLIENT CONFIGURATION PROPERTIES***]
  README.md: |
    [***README CONTENT***]
---
apiVersion: v1
kind: Secret
metadata:
  name: [***TLS SECRET NAME***]
```

```
  namespace: [***NAMESPACE***]
type: Opaque
stringData:
  ca.crt: |
    [***CA CERTIFICATE (PEM)***]
```

- *[***NAMESPACE***]* must be the same namespace as the Cloudera Surveyor release.
- The key names (external.properties, README.md, ca.crt) must match the secretRef.key values you declare in your values file when you add clientConfigs to the cluster. When using kubectl, the key name is the part before the = in each --from-file flag.
- You can add more files to a `Secret` by adding more --from-file flags or more keys under stringData. Each key becomes a separate file entry in clientConfigs.files.

Adding client configurations to a registered cluster

Declare clientConfigs in your values file for a registered Kafka cluster to populate the Client Configurations tab with files for end users to download.

Before you begin

You have created the Kubernetes `Secrets` that hold the file content in the Cloudera Surveyor release namespace. See Creating Kubernetes Secrets for client configuration files on page 13.

Procedure

1. Add a clientConfigs block under the cluster entry in clusterConfigs.clusters in your values file.

```
#...
clusterConfigs:
  clusters:
    - clusterName: "[***CLUSTER NAME***]"
      bootstrapServers: "[***BOOTSTRAP SERVERS***]"
      clientConfigs:
        files:
          - name: external.properties
            contentType: text/plain
            description: "Kafka client properties for the external liste
ner."
            secretRef:
              name: [***CLIENT CONFIG SECRET NAME***]
              key: external.properties
          - name: kafka-ca.crt
            contentType: application/x-pem-file
            description: "CA certificate for the Kafka broker."
            secretRef:
              name: [***TLS SECRET NAME***]
              key: ca.crt
          - name: README.md
            contentType: text/markdown
            description: "Usage instructions."
            secretRef:
              name: [***CLIENT CONFIG SECRET NAME***]
              key: README.md
```

- clientConfigs.files[*].name – The filename shown in the UI and used inside the zip archive. The Cloudera Surveyor Helm chart also uses this value as the file basename on the pod under /etc/surveyor/client-configs/*[***CLUSTER NAME***]*/. No separate path property is required.
- clientConfigs.files[*].contentType – The MIME type of the file. End users can preview files with a text/* or application/x-pem-file content type in the UI. Other types are available for download only.

- clientConfigs.files[*].description – Optional. A human-readable description shown alongside the filename in the UI.
- clientConfigs.files[*].secretRef.name – The name of the Kubernetes `Secret` in the Cloudera Surveyor release namespace that contains the file content.
- clientConfigs.files[*].secretRef.key – The key within the `Secret` whose value is the file content. Required.

2. Apply the changes with helm upgrade.

```
helm upgrade [***RELEASE NAME***] [***CHART***] \
  --namespace [***NAMESPACE***] \
  --values [***MY-VALUES.YAML***] \
  --reuse-values
```

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 --values when running helm upgrade. Additionally use the --reuse-values option to retain other properties.

Results

The **Client Configurations** tab shows the configured files for the cluster. End users can preview, copy, and download the files from the tab.

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 Helm chart creates a `ClusterIP` type Kubernetes `Service`, which is only reachable from within the Kubernetes cluster. To provide external access to the UI, you must configure an `Ingress` or deploy a `LoadBalancer` type `Service`.

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 values file (values.yaml).

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

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.

Configuring snapshots

Learn about snapshots and configuring snapshot behavior in Cloudera Surveyor. Snapshots control how frequently data is presented on the UI.

About snapshots

Cloudera Surveyor automatically collects snapshots of Kafka cluster data at configured intervals. These snapshots capture the current state of topics, partitions, consumer groups, and other cluster metadata. Most data presented on the UI is based on these snapshots.

Because of snapshots, the majority of changes in Kafka clusters only appear on the UI after the next scheduled snapshot is taken. For example, if snapshots occur every 10 minutes, a new topic created in Kafka might not be visible in the UI for up to 10 minutes.

The snapshot system provides multiple configuration settings to control data collection intervals, timeouts, and resource usage. Configuration enables you to fine-tune data collection behavior. For example, more frequent

snapshots provide fresher data but consume more resources, while less frequent snapshots reduce resource usage but may show older data.

Note: Broker and topic configuration data is an exception to the snapshot system. Configuration changes are retrieved on demand during page load rather than relying on snapshots. Consequently, neither the backend snapshot settings nor the UI refresh controls affect this data.

Configuration levels and time format

Most snapshot settings can be configured at two levels:

- **Global settings** – Apply to all clusters by default. Configured using surveyorConfig.surveyor.* properties.
- **Per-cluster overrides** – Override global settings for specific clusters. Configured using clusterConfigs.clusters[*].* properties.

Per-cluster settings take precedence over global settings, allowing you to customize behavior for individual clusters while maintaining sensible defaults for all others.

All duration and interval values throughout the snapshot configuration are specified in ISO-8601 format. For example, PT5M for 5 minutes, PT1H for 1 hour, and PT30S for 30 seconds.

Configuring snapshot intervals

Configure how frequently Cloudera Surveyor collects snapshots from registered Kafka clusters.

Snapshot intervals determine how frequently Cloudera Surveyor collects data from Kafka clusters. The snapshot interval has a direct effect on how fresh the data is that is presented on the UI.

Configure the snapshot interval globally for all clusters using surveyorConfig.surveyor.globalSnapshotInterval or on a per cluster basis using clusterConfigs.clusters[*].snapshotInterval. High-activity clusters might benefit from more frequent snapshots, while stable clusters can use longer intervals to reduce resource usage.

The following example configures a global snapshot interval of 10 minutes and configures per cluster overrides for a production and development cluster.

```
#...
surveyorConfig:
  surveyor:
    globalSnapshotInterval: PT10M

clusterConfigs:
  clusters:
    - clusterName: "production-cluster"
      bootstrapServers: "prod-kafka-1:9092,prod-kafka-2:9092"
      snapshotInterval: PT5M
    - clusterName: "development-cluster"
      bootstrapServers: "dev-kafka:9092"
      snapshotInterval: PT30M
    - clusterName: "staging-cluster"
      bootstrapServers: "staging-kafka:9092"
```

The production-cluster has a decreased interval for more frequent updates, the development-cluster has a much longer interval, while the staging-cluster has no overrides and uses the global default.

Configuring snapshot reliability settings

Configure advanced snapshot settings to optimize reliability and performance for clusters with specific network conditions, resource constraints, or availability requirements.

Procedure

Beyond basic snapshot intervals, Cloudera Surveyor provides additional properties to handle various operational scenarios. Use these settings to fine-tune snapshotting behavior. Timeouts prevent hanging on slow clusters, time-to-live (TTL) settings maintain data availability during temporary failures, and resource management settings control system load.

- Configure snapshot timeout settings with surveyorConfig.surveyor.globalSnapshotTimeout globally or with clusterConfigs.clusters[*].snapshotTimeout per cluster to handle slow or unresponsive clusters.

  Snapshot timeouts prevent Cloudera Surveyor from waiting indefinitely for slow clusters. Per-cluster overrides are useful for clusters that consistently need more time. For example:

  ```
  #...
  surveyorConfig:
    surveyor:
      globalSnapshotTimeout: PT2M

  clusterConfigs:
    clusters:
      - clusterName: "slow-cluster"
        bootstrapServers: "slow-kafka:9092"
        snapshotTimeout: PT5M
  ```

- Configure snapshot TTL settings with surveyorConfig.surveyor.globalSnapshotTtl globally or with clusterConfigs.clusters[*].snapshotTtl per cluster to handle intermittent failures.

  TTL settings determine how long to keep the last successful snapshot when subsequent snapshots fail. Per-cluster overrides are useful for clusters with intermittent connectivity issues. For example:

  ```
  #...
  surveyorConfig:
    surveyor:
      globalSnapshotTtl: PT15M

  clusterConfigs:
    clusters:
      - clusterName: "unstable-cluster"
        bootstrapServers: "unstable-kafka:9092"
        snapshotTtl: PT30M
  ```

- Configure global resource management settings with surveyorConfig.surveyor.maxGlobalSnapshotParallelism and with surveyorConfig.surveyor.snapshotMaxJitter to control system-wide snapshot processing.

  These properties control system-wide resources and apply to all clusters. They are global-only and cannot be configured per cluster. The maxGlobalSnapshotParallelism property limits the number of threads for processing snapshots across all clusters. The snapshotMaxJitter property adds an initial delay to distribute load when managing many clusters. For example:

  ```
  #...
  surveyorConfig:
    surveyor:
      maxGlobalSnapshotParallelism: 4
      snapshotMaxJitter: PT30S
  ```

Related Information

Client pools and client configuration hierarchy

Cloudera Surveyor Helm chart configuration reference