

Installation

Date published: 2024-06-11

Date modified: 2026-07-31



Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Installation overview.....	4
Installing Strimzi with Helm.....	6
Installing Strimzi in an internet environment.....	7
Installing Strimzi in an air-gapped environment.....	9
Installing Strimzi Drain Cleaner with Helm.....	12
Installing Strimzi Drain Cleaner in an internet environment.....	12
Installing Strimzi Drain Cleaner in an air-gapped environment.....	14
Installing Cloudera Surveyor for Apache Kafka with Helm.....	18
Installing Cloudera Surveyor in an internet environment.....	18
Installing Cloudera Surveyor in an air-gapped environment.....	25
Installing Cloudera Surveyor for evaluation.....	33
Installing Schema Registry with Helm.....	36
Installing Schema Registry in an internet environment.....	36
Installing Schema Registry in an air-gapped environment.....	42
Installing Schema Registry for evaluation.....	48
Installing Strimzi on Taikun CloudWorks [Technical Preview].....	51
Importing the Cloudera Kafka Operator repository and adding Strimzi to a catalog.....	51
Installing Strimzi.....	52
Installing Cloudera Surveyor for Apache Kafka on Taikun CloudWorks [Technical Preview].....	54
Importing the Cloudera Surveyor repository and adding Cloudera Surveyor to a catalog.....	54
Installing Cloudera Surveyor.....	55

Installation overview

Get started with installing Cloudera Streams Messaging Operator for Kubernetes. Learn about installable components, installation methods, and where installation artifacts are hosted.

Installable components

Cloudera Streams Messaging Operator for Kubernetes includes multiple installable components:

- **Strimzi** – Deploys and manages Kafka and Kafka Connect clusters on Kubernetes.
- **Strimzi Drain Cleaner** – Utility for safely moving Strimzi-managed Kafka Pods off Kubernetes nodes during maintenance or upgrades.
- **Cloudera Surveyor** – A UI application for monitoring and managing Kafka clusters.
- **Schema Registry** – Provides centralized schema storage and validation.

These components are largely independent and are installed with separate Helm charts. You can install any subset depending on your use case and operational objectives. For the complete operational experience, Cloudera recommends installing all components. Strimzi Drain Cleaner is installed with its own Helm chart but requires an existing Strimzi deployment. Cloudera strongly recommends installing it whenever you install Strimzi.

Installation methods

The following installation methods are available:

- **Installing with Helm** – The default installation method that works on any supported Kubernetes environment. Installation involves logging in to the Cloudera Docker registry, creating **Secrets** for registry credentials and licensing, and installing using the `helm install` command.
- **Installing on Taikun CloudWorks** – If you are on Taikun CloudWorks, use the Taikun CloudWorks UI for installation. Installation involves importing repositories, adding components to catalogs, configuring parameters, and then installing through the UI.

Installation artifacts and artifact locations

Cloudera Streams Messaging Operator for Kubernetes ships with various installation artifacts. These artifacts are hosted at two locations, the Cloudera Docker registry and the Cloudera Archive.

Both the Cloudera Docker registry and the Cloudera Archive require Cloudera credentials (username and password) for access. Credentials are provided to you as part of your license and subscription agreement. You can access both the registry and the archive using the same credentials.

Cloudera Docker registry – container.repository.cloudera.com

The Docker registry hosts the Helm chart as well as all Docker images required in a Cloudera Streams Messaging Operator for Kubernetes environment. This includes Strimzi, Kafka, Strimzi Drain Cleaner, Cloudera Surveyor, as well as Schema Registry artifacts.

Table 1: Strimzi and Kafka artifacts on the Cloudera Docker registry

Artifact	Location	Description
Strimzi Docker image	<code>container.repository.cloudera.com/cloudera/kafka-operator:1.0.1-csmop-1.7.0-b248</code>	Docker image used for deploying Strimzi and its components.
Kafka Docker image	<code>container.repository.cloudera.com/cloudera/kafka:1.0.1-csmop-1.7.0-b248-kafka-4.2.0-csmop-1.7</code>	Docker image used for deploying Kafka and related components.
Strimzi Cluster Operator Helm chart	<code>oci://container.repository.cloudera.com/cloudera-helm/csm-operator/strimzi-kafka-operator:1.7.0-b248</code>	Helm chart used to install the Strimzi Cluster Operator with helm install.

Artifact	Location	Description
Buildah Docker image	container.repository.cloudera.com/cloudera/buildah:1.0.1-csmop-1.7.0-b248	Docker image used by Strimzi to automatically build new Kafka images that include custom Kafka Connect connector plugins .

Table 2: Strimzi Drain Cleaner artifacts on the Cloudera Docker registry

Artifact	Location	Description
Strimzi Drain Cleaner Docker image	container.repository.cloudera.com/cloudera/drain-cleaner:1.6.0-csmop-1.7.0-b248	Docker image for the Strimzi Drain Cleaner validating webhook.
Strimzi Drain Cleaner Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/strimzi-drain-cleaner:1.7.0-b248	Helm chart used to install Strimzi Drain Cleaner with helm install.

Table 3: Cloudera Surveyor artifacts on the Cloudera Docker registry

Artifact	Location	Description
Cloudera Surveyor server Docker image	container.repository.cloudera.com/cloudera/surveyor:0.2.0-csmop-1.7.0-b248	Docker image used for deploying Cloudera Surveyor.
Cloudera Surveyor UI application Docker image	container.repository.cloudera.com/cloudera/surveyor-app:0.2.0-csmop-1.7.0-b248	Docker image used for deploying Cloudera Surveyor.
Cloudera Surveyor Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surveyor:1.7.0-b248	Helm chart used to install Cloudera Surveyor with helm-install.

Table 4: Schema Registry artifacts on the Cloudera Docker registry

Artifact	Location	Description
Schema Registry Docker image	container.repository.cloudera.com/cloudera/schema-registry:0.10.0-csmop-1.7.0-b248	Docker image used for deploying Schema Registry.
Schema Registry Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/schema-registry:1.7.0-b248	Helm chart used to install Schema Registry with helm-install.



Note: The images are built for linux/arm64 and linux/amd64 architectures.

Cloudera Archive – archive.cloudera.com/p/csm-operator/

The Cloudera Archive hosts various installation artifacts including Helm charts, configuration examples, a YAML file containing all CRDs, diagnostic tools, and the maven artifacts.

Accessing the Cloudera Archive and the artifacts it hosts is not necessary to complete installation. All artifacts on the archive are supplemental resources. The following table collects the Cloudera Streams Messaging Operator for Kubernetes directories located in the archive with an overview of what artifacts they contain and how you can use them.

Table 5: Cloudera Streams Messaging Operator for Kubernetes directories on the Cloudera Archive

Archive Directory	Description
https://archive.cloudera.com/p/csm-operator/1.7/charts/	The charts directory contains the Helm charts for components. These are the same charts that are available on the Docker registry. Cloudera recommends that whenever possible you install with the charts hosted on the registry. The charts on the archive are provided in case you cannot access the registry or want to download the chart using a browser.
https://archive.cloudera.com/p/csm-operator/1.7/examples/	The examples directory includes various examples of resource configuration files. You can use these to quickly deploy Kafka and other components in Kubernetes following installation.
https://archive.cloudera.com/p/csm-operator/1.7/install/	The install directory contains a single YAML file that collects all Strimzi Cluster Operator CRDs. The purpose of this file is twofold. One, the CRDs are rich in comments. Reviewing them can help you better understand how Kafka is deployed and managed with Strimzi in Cloudera Streams Messaging Operator for Kubernetes. It is a supplemental resource to the documentation. Two, this file is used to upgrade CRDs during upgrades. The CRDs are also included in the Strimzi Cluster Operator Helm chart, and Helm will automatically install the necessary CRDs to Kubernetes. You do not need to install them separately with the file hosted on the archive.
https://archive.cloudera.com/p/csm-operator/1.7/maven-repository/	The maven artifacts can be used to develop your own applications or tools for use with Cloudera Streams Messaging Operator for Kubernetes.
https://archive.cloudera.com/p/csm-operator/1.7/tools/	The tools directory contains command line tools that you use to collect diagnostic information and to troubleshoot cluster issues.

Installing Strimzi with Helm

Learn how to install Strimzi in Cloudera Streams Messaging Operator for Kubernetes with Helm. Installing Strimzi installs the applications and resources that enable you to deploy and manage Kafka in Kubernetes.

Strimzi is installed in your Kubernetes cluster with the Strimzi Cluster Operator Helm chart using the `helm install` command. When you install the chart, Helm installs the Strimzi Custom Resource Definitions (CRDs) included in Cloudera Streams Messaging Operator for Kubernetes and deploys the Strimzi Cluster Operator, which is an operator application that manages and monitors Kafka and related components. Additionally, other cluster resources and applications required for managing Kafka are also installed.

Installing Strimzi does not create or deploy a Kafka cluster. Kafka clusters are created following the installation by deploying `Kafka` and `KafkaNodePool` resources in the Kubernetes cluster with `kubectl` or `oc`.

Cloudera recommends that you install Strimzi once per Kubernetes cluster. Some resources are cluster-wide, which can cause issues if Strimzi is installed multiple times on the same cluster.

By default, the Strimzi Cluster Operator (deployed with installation) watches and manages the Kafka clusters that are deployed in the same namespace as the Strimzi Cluster Operator. However, you can configure it to watch any namespace. This allows you to manage multiple Kafka clusters deployed in different namespaces using a single installation.

Installation instructions are provided for the following scenarios.

- **Installing in an internet environment** – Follow these steps to install Strimzi in a Kubernetes cluster with internet access.
- **Installing in an air-gapped environment** – Follow these steps to install Strimzi in a Kubernetes cluster without internet access or if you want to install from a self-hosted registry.

Installing Strimzi in an internet environment

Complete these steps to install Strimzi if your Kubernetes cluster has internet access.

Before you begin

- Your Kubernetes environment meets requirements listed in [System requirements](#).
- Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- If you are planning to watch and manage more than 20 Kafka clusters with a single installation, you must increase the memory and heap allocated to the Strimzi Cluster Operator. You can specify memory configuration in your helm install command. For more information, see [Increasing Cluster Operator memory](#).

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Strimzi. Use the namespace you create in all installation steps that follow.

2. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

3. Create a Kubernetes `Secret` containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
-s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.
- Replace `[***USERNAME***]` with your Cloudera username.
- Enter your Cloudera password when prompted.



Important: The `Secret` containing your credentials must exist in the namespace where you install Strimzi as well as all namespaces where you deploy Kafka or Kafka Connect clusters. Cloudera recommends that you create the `Secret` in all required namespaces now if you know what namespaces you will be using to deploy Kafka or Kafka Connect.

4. Install Strimzi with helm install.

```
helm install strimzi-cluster-operator \
  --namespace [***NAMESPACE***] \
  --set 'image.imagePullSecrets[0].name=[***REGISTRY CREDENTIALS SECRET***]' \
  --set-file clouderaLicense.fileContent=[***PATH TO LICENSE FILE***] \
  --set watchAnyNamespace=true \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/strimzi-kafka-operator \
  --version 1.7.0-b248
```

- The string `strimzi-cluster-operator` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- `imagePullSecrets` specifies what secret is used to pull images from the Cloudera registry. Setting this property is mandatory, otherwise, Helm cannot pull the necessary images from the Cloudera Docker registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the secret you created in Step 3 on page 7.
- `clouderaLicense.fileContent` is used to register your license. If this property is set, a secret is generated that contains the license you specify. Setting this property is mandatory. The Strimzi Cluster Operator will not function without a valid license. Ensure that you replace `[***PATH TO LICENSE FILE***]` with the full path to your Cloudera license file.
- You can use `--set` to set various other properties of the Helm chart. This enables you to customize your installation. For example, Cloudera recommends that you configure the Cluster Operator to watch all namespaces, this is configured by setting `watchAnyNamespace` to `true`. Alternatively, you can configure a list of specific namespaces to watch using `watchNamespaces`.

5. Verify your installation

This is done by listing the deployments and pods in your namespace. If installation is successful, you should see a `strimzi-cluster-operator` deployment and pod present in the cluster.

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

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
#...				
strimzi-cluster-operator	1/1	1	1	13m

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

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
#...				
strimzi-cluster-operator	1/1	1	1	13m

6. Access supplemental resources available on the Cloudera Archive.

Supplemental resources available on the Cloudera Archive include various example files, diagnostic tools, and more. You can use these resources to quickly deploy Kafka clusters and to gain a better understanding of Strimzi and Cloudera Streams Messaging Operator for Kubernetes.

What to do next

- Deploy a Kafka cluster, see [Deploying Kafka](#).

- Set up Prometheus for monitoring, see [Configuring Kafka for Prometheus monitoring](#) and [Monitoring with Prometheus](#).

Related Information

[Cloudera Archive](#)

Installing Strimzi in an air-gapped environment

Complete these steps to install Strimzi if your Kubernetes cluster does not have internet access or if you want to install from a self-hosted registry.

Before you begin

- Your Kubernetes environment meets requirements listed in [System requirements](#).
- A self-hosted Docker registry is required. Your registry must be accessible by your Kubernetes cluster.
- A machine with Internet connectivity is required. While the Kubernetes cluster does not need internet access, you will need a machine to pull the images from the Cloudera Docker registry.
- Access to docker or equivalent utility that you can use to pull and push images is required. The following steps use docker. Replace commands where necessary.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- If you are planning to watch and manage more than 20 Kafka clusters with a single installation, you must increase the memory and heap allocated to the Strimzi Cluster Operator. You can specify memory configuration in your helm install command. For more information, see [Increasing Cluster Operator memory](#).

Procedure

1. Copy the following installation artifacts to your self-hosted registry.

Table 6: Strimzi and Kafka artifacts on the Cloudera Docker registry

Artifact	Location	Description
Strimzi Docker image	container.repository.cloudera.com/cloudera/kafka-operator:1.0.1-csmop-1.7.0-b248	Docker image used for deploying Strimzi and its components.
Kafka Docker image	container.repository.cloudera.com/cloudera/kafka:1.0.1-csmop-1.7.0-b248-kafka-4.2.0-csmop-1.7	Docker image used for deploying Kafka and related components.
Strimzi Cluster Operator Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/strimzi-kafka-operator:1.7.0-b248	Helm chart used to install the Strimzi Cluster Operator with helm install.
Buildah Docker image	container.repository.cloudera.com/cloudera/buildah:1.0.1-csmop-1.7.0-b248	Docker image used by Strimzi to automatically build new Kafka images that include custom Kafka Connect connector plugins .



Note: The images are built for linux/arm64 and linux/amd64 architectures.

This step involves pulling the artifacts from the Cloudera Docker registry, retagging them, and then pushing them to your self-hosted registry. The exact steps you need to carry out depend on your environment and how your registry is set up. The following substeps demonstrate the basic workflow using docker and helm.



Tip: If your registry uses a non-default port, you might need to specify the registry port in helm as well as docker commands.

- a) Log in to the Cloudera Docker registry with both docker and helm.
Provide your Cloudera credentials when prompted.

```
docker login container.repository.cloudera.com
```

```
helm registry login container.repository.cloudera.com
```

- b) Log in to your self-hosted registry with both docker and helm.
Provide credentials when prompted.

```
docker login [***YOUR REGISTRY***]
```

```
helm registry login [***YOUR REGISTRY***]
```

- c) Pull the Docker images from the Cloudera Docker registry.

```
docker pull \
  --platform [***PLATFORM/ARCHITECTURE***] \
  container.repository.cloudera.com/cloudera/[***IMAGE
  NAME***]:[***VERSION***]
```

- d) Pull the Strimzi Cluster Operator Helm chart.

```
helm pull \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/str
  imzi-kafka-operator \
  --version 1.7.0-b248
```

- e) Retag the Docker images you pulled so that they contain the address of your registry.

```
docker tag \
  [***ORIGINAL IMAGE TAG***] \
  [***YOUR REGISTRY***]/cloudera/[***IMAGE NAME***]:[***VERSION***]
```

- f) Push the images and chart to your self-hosted registry.

```
docker push \
  [***YOUR REGISTRY***]/cloudera/[***IMAGE NAME***]:[***VERSION***]
```

```
helm push \
  strimzi-kafka-operator-1.7.0-b248.tgz \
  oci://[***YOUR REGISTRY***]/cloudera-helm/csm-operator/
```

2. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Strimzi. Use the namespace you create in all installation steps that follow.

3. Create a Kubernetes Secret containing credentials for your self-hosted registry.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***]
\
  --namespace [***NAMESPACE***] \
  --docker-server [***YOUR REGISTRY***] \
  --docker-username [***USERNAME***] \
```

```
--docker-password "$(echo -n 'Enter your password: ' >&2; read -s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `***REGISTRY CREDENTIALS SECRET***`. You will need to specify the name in a later step.
- Replace `***USERNAME***` with a username valid for your self-hosted registry.
- Enter the password for the user when prompted.



Important: The `Secret` containing your credentials must exist in the namespace where you install Strimzi as well as all namespaces where you deploy Kafka or Kafka Connect clusters. Cloudera recommends that you create the `Secret` in all required namespaces now if you know what namespaces you will be using to deploy Kafka or Kafka Connect.

4. Install Strimzi with helm install.

```
helm install strimzi-cluster-operator \
  --namespace [***NAMESPACE***] \
  --set 'image.imagePullSecrets[0].name=[***REGISTRY CREDENTIALS SECRET***]' \
  --set defaultImageRegistry=[***YOUR REGISTRY***] \
  --set-file clouderaLicense.fileContent=[***PATH TO LICENSE FILE***] \
  oci://[***YOUR REGISTRY***]/cloudera-helm/csm-operator/strimzi-kafka-operator \
  --version 1.7.0-b248 \
  --set watchAnyNamespace=true
```

- The string `strimzi-cluster-operator` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- `imagePullSecrets` specifies what secret is used to pull images from the specified registry. Ensure that you replace `***REGISTRY CREDENTIALS SECRET***` with the name of the secret you created in Step 3 on page 10.
- `clouderaLicense.fileContent` is used to register your license. If this property is set, a secret is generated that contains the license you specify. Setting this property is mandatory. The Strimzi Cluster Operator will not function without a valid license. Ensure that you replace `***PATH TO LICENSE FILE***` with the full path to your Cloudera license file.
- You can use `--set` to set various other properties of the Helm chart. This enables you to customize your installation. For example, Cloudera recommends that you configure the Cluster Operator to watch all namespaces, this is configured by setting `watchAnyNamespace` to `true`. Alternatively, you can configure a list of specific namespaces to watch using `watchNamespaces`.

5. Verify your installation

This is done by listing the deployments and pods in your namespace. If installation is successful, you should see a `strimzi-cluster-operator` deployment and pod present in the cluster.

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

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
#...				
strimzi-cluster-operator	1/1	1	1	13m

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

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
#...				
strimzi-cluster-operator	1/1	1	1	13m

6. Access supplemental resources available on the Cloudera Archive.

Supplemental resources available on the Cloudera Archive include various example files, diagnostic tools, and more. You can use these resources to quickly deploy Kafka clusters and to gain a better understanding of Strimzi and Cloudera Streams Messaging Operator for Kubernetes.

What to do next

- Deploy a Kafka cluster, see [Deploying Kafka](#).
- Set up Prometheus for monitoring, see [Configuring Kafka for Prometheus monitoring](#) and [Monitoring with Prometheus](#).

Related Information

[Cloudera Archive](#)

Installing Strimzi Drain Cleaner with Helm

Learn how to install Strimzi Drain Cleaner in Cloudera Streams Messaging Operator for Kubernetes with Helm. Strimzi Drain Cleaner coordinates safe rolling restarts of Strimzi-managed Kafka Pods when Kubernetes nodes are drained.

Strimzi Drain Cleaner is installed in your Kubernetes cluster with the Strimzi Drain Cleaner Helm chart using the helm install command. When you install the chart, Helm deploys the validating admission webhook and supporting resources.

Strimzi Drain Cleaner is installed with a separate Helm chart but requires an existing Strimzi deployment. Strimzi Drain Cleaner is most useful after you deploy at least one Kafka cluster. Cloudera recommends doing so before or soon after installation.

Install Strimzi Drain Cleaner in the same Kubernetes cluster where Strimzi and your Kafka clusters run. Only a single installation is supported per Kubernetes cluster. This is because the chart registers a cluster-scoped `ValidatingWebhookConfiguration`.

With default chart values, cert-manager must be installed and operational in your Kubernetes cluster. The chart creates cert-manager `Certificate` and `Issuer` resources automatically during installation.

Installing Strimzi Drain Cleaner does not require a Cloudera license.

Strimzi Drain Cleaner is deployed to the `strimzi-drain-cleaner` namespace. The installation steps create that namespace before you run `helm install` so you can complete required preparation before the chart deploys Strimzi Drain Cleaner.



Important: Installation using YAML or installation files is not supported in Cloudera Streams Messaging Operator for Kubernetes. Install Strimzi Drain Cleaner with Helm only.

Installation instructions are provided for the following scenarios.

- **Installing in an internet environment** – Follow these steps to install Strimzi Drain Cleaner in a Kubernetes cluster with internet access.
- **Installing in an air-gapped environment** – Follow these steps to install Strimzi Drain Cleaner in a Kubernetes cluster without internet access or if you want to install from a self-hosted registry.

Installing Strimzi Drain Cleaner in an internet environment

Complete these steps to install Strimzi Drain Cleaner if your Kubernetes cluster has internet access.

Before you begin

- Your Kubernetes environment meets requirements listed in [System requirements](#).

- Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- Ensure that the Strimzi Cluster Operator is installed and running. See [Installation](#).
- cert-manager is installed and operational in your Kubernetes cluster. With default chart values, the Strimzi Drain Cleaner Helm chart creates cert-manager `Certificate` and `Issuer` resources during installation.
- Review the [Strimzi Drain Cleaner Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

Procedure

1. Create the `strimzi-drain-cleaner` namespace in your Kubernetes cluster.

```
kubectl create namespace strimzi-drain-cleaner
```

2. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

3. Create a Kubernetes `Secret` containing your Cloudera credentials in the `strimzi-drain-cleaner` namespace.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace strimzi-drain-cleaner \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' && read
  -s password; echo && echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You specify the name in a later step.
 - Replace `[***USERNAME***]` with your Cloudera username.
 - Enter your Cloudera password when prompted.
4. Install Strimzi Drain Cleaner with `helm install`.

```
helm install strimzi-drain-cleaner \
  --namespace strimzi-drain-cleaner \
  --set namespace.create=false \
  --set 'image.imagePullSecrets[0].name=[***REGISTRY CREDENTIALS
  SECRET***]' \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/str
  imzi-drain-cleaner \
  --version 1.7.0-b248
```

- The string `strimzi-drain-cleaner` is the Helm release name. You can choose a different name, but Cloudera recommends using `strimzi-drain-cleaner` so the Helm release name matches the Kubernetes resources the chart creates.
- `namespace.create=false` prevents the chart from creating the namespace because you created `strimzi-drain-cleaner` in Step 1.
- `image.imagePullSecrets` specifies what secret is used to pull images from the Cloudera registry. Setting this property is mandatory. Without it, Helm cannot pull the Strimzi Drain Cleaner image from the Cloudera Docker registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the secret you created in Step 3 on page 13.

5. Verify your installation.

If installation is successful, you should see a `strimzi-drain-cleaner` Helm release, Deployment, and Pods in the `strimzi-drain-cleaner` namespace. In addition a `ValidatingWebhookConfiguration` should exist as well.

Verify the Helm release with `helm list`:

```
helm list --namespace strimzi-drain-cleaner
```

NAME	NAMESPACE	STATUS	CHART	REVISION	UPDATED	APP VERSION
strimzi-drain-cleaner	strimzi-drain-cleaner	16:48:28.661181 +0200 CEST	deployed	1	2026-07-07	2026-07-07
			1.6.0-csmop-1.7.0-b248			strimzi-drain-cleaner-1.7.0-b248

Verify Deployment with `kubectl get` deployments:

```
kubectl get deployments --namespace strimzi-drain-cleaner
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
strimzi-drain-cleaner	2/2	2	2	95s

Verify Pods with `kubectl get` pods:

```
kubectl get pods --namespace strimzi-drain-cleaner
```

NAME	READY	STATUS	RESTARTS	AGE
strimzi-drain-cleaner-776cc6d7b-ff2d4	1/1	Running	0	102s
strimzi-drain-cleaner-776cc6d7b-qj9fv	1/1	Running	0	102s

Verify that the cluster-scoped `ValidatingWebhookConfiguration` resource exists:

```
kubectl get validatingwebhookconfiguration strimzi-drain-cleaner
```

NAME	WEBHOOKS	AGE
strimzi-drain-cleaner	1	4m23s

What to do next

After installation, Strimzi Drain Cleaner runs continuously in the background. It requires no day-to-day operator action and responds automatically when Strimzi-managed Kafka Pods are evicted.

Related Information

[Strimzi Drain Cleaner](#)

[Strimzi Drain Cleaner Helm reference](#)

[Deploying Kafka](#)

Installing Strimzi Drain Cleaner in an air-gapped environment

Complete these steps to install Strimzi Drain Cleaner if your Kubernetes cluster does not have internet access or if you want to install from a self-hosted registry.

Before you begin

- Your Kubernetes environment meets requirements listed in [System requirements](#).

- A self-hosted Docker registry is required. Your registry must be accessible by your Kubernetes cluster.
- A machine with Internet connectivity is required. While the Kubernetes cluster does not need internet access, you will need a machine to pull the images from the Cloudera Docker registry.
- Access to docker or equivalent utility that you can use to pull and push images is required. The following steps use docker. Replace commands where necessary.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- Ensure that the Strimzi Cluster Operator is installed and running. See [Installation](#).
- cert-manager is installed and operational in your Kubernetes cluster. With default chart values, the Strimzi Drain Cleaner Helm chart creates cert-manager `Certificate` and `Issuer` resources during installation.
- Review the [Strimzi Drain Cleaner Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

Procedure

1. Copy the following installation artifacts to your self-hosted registry.

Table 7: Strimzi Drain Cleaner artifacts on the Cloudera Docker registry

Artifact	Location	Description
Strimzi Drain Cleaner Docker image	container.repository.cloudera.com/cloudera/drain-cleaner:1.6.0-csmop-1.7.0-b248	Docker image for the Strimzi Drain Cleaner validating webhook.
Strimzi Drain Cleaner Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/strimzi-drain-cleaner:1.7.0-b248	Helm chart used to install Strimzi Drain Cleaner with helm install.



Note: The images are built for linux/arm64 and linux/amd64 architectures.

This step involves pulling the artifacts from the Cloudera Docker registry, retagging them, and then pushing them to your self-hosted registry. The exact steps you need to carry out depend on your environment and how your registry is set up. The following substeps demonstrate the basic workflow using docker and helm.



Tip: If your registry uses a non-default port, you might need to specify the registry port in helm as well as docker commands.

- a) Log in to the Cloudera Docker registry with both docker and helm.
Provide your Cloudera credentials when prompted.

```
docker login container.repository.cloudera.com
```

```
helm registry login container.repository.cloudera.com
```

- b) Log in to your self-hosted registry with both docker and helm.
Provide credentials when prompted.

```
docker login [***YOUR REGISTRY***]
```

```
helm registry login [***YOUR REGISTRY***]
```

- c) Pull the Strimzi Drain Cleaner Docker image from the Cloudera Docker registry.

```
docker pull \
  --platform [***PLATFORM/ARCHITECTURE***] \
```

```
container.repository.cloudera.com/cloudera/drain-cleaner:1.6.0-
csmop-1.7.0-b248
```

- d) Pull the Strimzi Drain Cleaner Helm chart.

```
helm pull \
oci://container.repository.cloudera.com/cloudera-helm/csm-operator/str
imzi-drain-cleaner \
--version 1.7.0-b248
```

- e) Retag the Strimzi Drain Cleaner Docker image you pulled so that it contains the address of your registry.

```
docker tag \
container.repository.cloudera.com/cloudera/drain-cleaner:1.6.0-
csmop-1.7.0-b248 \
[***YOUR_REGISTRY**]/cloudera/drain-cleaner:1.6.0-csmop-1.7.0-b248
```

- f) Push the Strimzi Drain Cleaner image and chart to your self-hosted registry.

```
docker push \
[***YOUR_REGISTRY**]/cloudera/drain-cleaner:1.6.0-csmop-1.7.0-b248
```

```
helm push \
strimzi-drain-cleaner-1.7.0-b248.tgz \
oci://[***YOUR_REGISTRY**]/cloudera-helm/csm-operator/
```

2. Create the strimzi-drain-cleaner namespace in your Kubernetes cluster.

```
kubectl create namespace strimzi-drain-cleaner
```

3. Create a Kubernetes Secret containing credentials for your self-hosted registry in the strimzi-drain-cleaner namespace.

```
kubectl create secret docker-registry [***REGISTRY_CREDENTIALS_SECRET**]
\
--namespace strimzi-drain-cleaner \
--docker-server [***YOUR_REGISTRY**] \
--docker-username [***USERNAME**] \
--docker-password "$(echo -n 'Enter your password: ' >&2; read -s passw
ord; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY_CREDENTIALS_SECRET**]`. You specify the name in a later step.
- Replace `[***YOUR_REGISTRY**]` and `[***USERNAME**]` with values valid for your self-hosted registry.
- Enter the password for the user when prompted.

4. Install Strimzi Drain Cleaner with helm install.

```
helm install strimzi-drain-cleaner \
--namespace strimzi-drain-cleaner \
--set namespace.create=false \
--set 'image.imagePullSecrets[0].name=[***REGISTRY_CREDENTIALS
SECRET**]' \
--set image.registry=[***YOUR_REGISTRY**] \
oci://[***YOUR_REGISTRY**]/cloudera-helm/csm-operator/strimzi-drain-cl
eaner \
--version 1.7.0-b248
```

- The string strimzi-drain-cleaner is the Helm release name. You can choose a different name, but Cloudera recommends using strimzi-drain-cleaner so the Helm release name matches the Kubernetes resources the chart creates.

- `namespace.create=false` prevents the chart from creating the namespace because you created `strimzi-drain-cleaner` in the previous step.
- `image.imagePullSecrets` specifies what secret is used to pull images from your self-hosted registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the secret you created in [Step 3](#) on page 16.
- `image.registry` must point to your self-hosted registry so that the chart pulls the retagged Strimzi Drain Cleaner image.

5. Verify your installation.

If installation is successful, you should see a `strimzi-drain-cleaner` Helm release, Deployment, and Pods in the `strimzi-drain-cleaner` namespace. In addition a `ValidatingWebhookConfiguration` should exist as well.

Verify the Helm release with `helm list`:

```
helm list --namespace strimzi-drain-cleaner
```

NAME	NAMESPACE	STATUS	CHART	REVISION	UPDATED	APP VERSION
strimzi-drain-cleaner	strimzi-drain-cleaner	16:48:28.661181 +0200 CEST	deployed	1	2026-07-07	2026-07-07
			1.6.0-csmop-1.7.0-b248			

Verify Deployment with `kubectl get` deployments:

```
kubectl get deployments --namespace strimzi-drain-cleaner
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
strimzi-drain-cleaner	2/2	2	2	95s

Verify Pods with `kubectl get` pods:

```
kubectl get pods --namespace strimzi-drain-cleaner
```

NAME	READY	STATUS	RESTARTS	AGE
strimzi-drain-cleaner-776cc6d7b-ff2d4	1/1	Running	0	102s
strimzi-drain-cleaner-776cc6d7b-qj9fv	1/1	Running	0	102s

Verify that the cluster-scoped `ValidatingWebhookConfiguration` resource exists:

```
kubectl get validatingwebhookconfiguration strimzi-drain-cleaner
```

NAME	WEBHOOKS	AGE
strimzi-drain-cleaner	1	4m23s

What to do next

After installation, Strimzi Drain Cleaner runs continuously in the background. It requires no day-to-day operator action and responds automatically when Strimzi-managed Kafka Pods are evicted.

Related Information

[Strimzi Drain Cleaner](#)

[Strimzi Drain Cleaner Helm reference](#)

[Deploying Kafka](#)

Installing Cloudera Surveyor for Apache Kafka with Helm

Learn how to install Cloudera Surveyor in Cloudera Streams Messaging Operator for Kubernetes with Helm. Cloudera Surveyor is a UI application that enables you to manage and monitor Kafka clusters.

Cloudera Surveyor is installed in your Kubernetes cluster with the Cloudera Surveyor Helm chart using the `helm install` command. When you install the chart, Helm deploys an instance of Cloudera Surveyor, which enables you to manage and monitor your Kafka clusters through a UI interface.

During installation, you configure Cloudera Surveyor using a custom values file (`values.yaml`) passed to the Helm chart with the `--values (-f)` option. This file contains properties for configuring Cloudera Surveyor itself as well as Kafka cluster-specific settings that define which Kafka clusters Cloudera Surveyor connects to. Additionally, some properties are configured with `--set` options.

Installation instructions are provided for the following scenarios.

- **Installing in an internet environment** – Follow these steps to install a fully secure instance of Cloudera Surveyor in a Kubernetes cluster with internet access.
- **Installing in an air-gapped environment** – Follow these steps to install a fully secure instance of Cloudera Surveyor in a Kubernetes cluster without internet access or if you want to install from a self-hosted registry.
- **Installing for evaluation** – Follow these steps to install an unsecure instance of Cloudera Surveyor for development or proof of concept purposes.

Installing Cloudera Surveyor in an internet environment

Complete these steps to install Cloudera Surveyor if your Kubernetes cluster has internet access. These steps install a fully secure instance of Cloudera Surveyor that has authentication, authorization, and channel encryption configured. The example configuration also demonstrates how you can connect a secure Kafka cluster to Cloudera Surveyor.

Before you begin

- General prerequisites:
 - Your Kubernetes environment meets requirements listed in [System requirements](#).
 - Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
 - You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
 - You have access to a valid Cloudera license.
 - Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- Prerequisites for channel encryption (TLS):
 - An Ingress controller is installed in your Kubernetes cluster. These steps use the [Ingress-NGinx controller](#).
 - Optional: cert-manager is installed in your Kubernetes cluster.
- Prerequisites for LDAP authentication:
 - An LDAP server is available that has TLS enabled.
 - The server is accessible from the Kubernetes cluster where Cloudera Surveyor is deployed.
 - Entries containing usernames and passwords are located under a common base in the directory information tree. Passwords must be stored in the `userPassword` attribute in the user entries.
 - If deploying Cloudera Surveyor with FIPS mode enabled (`fipsMode: true`), you must manually generate and configure an authentication key as part of installation.

The key must be generated and saved to a file, then configured using the `--set-file` option in your helm install command. For more information, see [Configuring LDAP authentication](#).

- Prerequisites for authorization:
 - Ensure that Kafka ACLs are set up for your Kafka cluster. Cloudera Surveyor uses Kafka ACLs to provide authorization.

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Cloudera Surveyor. Use the namespace you create in all installation steps that follow.

2. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

3. Create a Kubernetes Secret containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
  -s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.
 - Replace `[***USERNAME***]` with your Cloudera username.
 - Enter your Cloudera password when prompted.
4. Create Secrets for sensitive Kafka client configuration values.

Cloudera Surveyor connects to Kafka clusters as any other Kafka client and requires a client configuration. If the Kafka cluster is secured, the client configuration will include sensitive property values. Cloudera recommends that you store sensitive values in Secrets, mount the Secrets to the Cloudera Surveyor Container, and reference the values in your configuration instead of hard-coding them.

Typically, you need to create two Secrets. One contains the Kafka cluster truststore and password, and the other contains a JAAS configuration. The Kafka truststore must be in JKS or PKCS12 format.

```
kubectl create secret generic [***KAFKA TRUSTSTORE SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***KAFKA TRUSTSTORE KEY***]=[***PATH TO TRUSTSTORE***] \
  --from-file=[***KAFKA TRUSTSTORE PASSWORD KEY***]=[***PATH TO TRUSTSTORE
  PASSWORD FILE***]
```

```
kubectl create secret generic [***KAFKA SASL.JAAS.CONFIG SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***KAFKA SASL.JAAS.CONFIG KEY***]=[***PATH TO KAFKA
  SASL.JAAS.CONFIG FILE***]
```

- Take note of the Secret names as well as the key names you configure. You will need to specify them in a later step.

- All key names like `***KAFKA TRUSTSTORE KEY***` or `***KAFKA SASL.JAAS.CONFIG KEY***` are arbitrary.
- `***PATH TO KAFKA SASL.JAAS.CONFIG FILE***` is a path to a file containing JAAS configuration similar to the following example:

```
org.apache.kafka.common.security.plain.PlainLoginModule required username="MY-USER" password="MY-PASSWORD";
```

The contents of this file are set in a later step as the value of the `sasl.jaas.config` Kafka client property for Cloudera Surveyor internal Kafka clients. Ensure that the format of the configuration is valid for the `sasl.jaas.config` property. That is, its a single line of configuration.

5. Prepare a certificate and private key for Cloudera Surveyor.

- If you have `cert-manager` available, create a `Certificate` resource. Take note of the `Secret` name you configure in `spec.secretname` of the `Certificate` resource, you will need to specify it in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. Take note of the `Secret` name, you will need to specify it in a later step.

This `Secret` is referred to as `***SURVEYOR TLS CERT SECRET***` in the following steps.

6. Prepare a certificate and private key for Ingress.

- If you have `cert-manager` available, the certificate and private key for `Ingress` are automatically requested by the `Ingress`. You only need to ensure that you have a valid `Issuer` available in `cert-manager`. You specify the name of the `Issuer` resource in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. Take note of the `Secret` name, you will need to specify it in a later step.

This `Secret` referred to as `***INGRESS TLS CERT SECRET***` in the following steps.

7. Set up resources for LDAP authentication.

- Generate a Java truststore (PKCS12 or JKS) containing the TLS certificate of the root Certificate Authority (CA) of the LDAP certificate chain.

```
keytool -import -trustcacerts -file [***LDAP ROOT CA***] \
  -keystore [***TRUSTSTORE NAME***] \
  -storepass [***TRUSTSTORE PASSWORD***] \
  -storetype PKCS12
```

- Create a `Secret` containing the truststore and the truststore password.

```
kubectl create secret generic [***LDAP TRUSTSTORE SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***TRUSTSTORE SECRET KEY***]=[***TRUSTSTORE NAME***] \
  --from-file=[***TRUSTSTORE PW SECRET KEY***]=[***PATH TO TRUSTSTORE PW FILE***]
```

Take note of `***LDAP TRUSTSTORE SECRET***`, `***TRUSTSTORE SECRET KEY***`, and `***TRUSTSTORE PW SECRET KEY***`. You will need to specify these names in a custom values file you create in a later step.

- Create a `Secret` containing your LDAP principal and password (bind credentials).

```
kubectl create secret generic [***LDAP CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-literal=principal="$(echo -n 'Enter principal: ' >&2; read -s principal; echo >&2; echo $principal)" --from-literal=password="$(echo -n 'Enter password: ' >&2; read -s password; echo >&2; echo $password)"
```

- Take note of `***LDAP CREDENTIALS SECRET***`. You will need to specify this name in a custom values file you create in a later step.

- Enter your principal and the password for the principal when prompted. Example principal:

```
cn=admin,dc=openldap-chart,dc=ldap
```

8. Prepare a custom values file (values.yaml).

The values file contains configuration for Cloudera Surveyor. This file specifies the Kafka clusters that Cloudera Surveyor connects to as well as various other configuration properties.

```
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/[***KAFKA TRUSTSTORE
SECRET***]/[***KAFKA TRUSTSTORE FILE***]"
        ssl.truststore.password: "\\${dir:/opt/secrets/[***KAFKA
TRUSTSTORE SECRET***]:[***KAFKA TRUSTSTORE PASSWORD FILE***]}"
        sasl.jaas.config: "\\${dir:/opt/secrets/[***KAFKA SASL.JAAS.CONFIG
SECRET***]:[***KAFKA SASL.JAAS.CONFIG FILE***]}"
        adminOperationTimeout: PT1M
        authorization:
          enabled: true
      secretsToMount:
        - create: false
          secretRef: [***KAFKA TRUSTSTORE SECRET***]
          items:
            - key: [***KAFKA TRUSTSTORE KEY***]
              path: [***KAFKA TRUSTSTORE FILE***]
            - key: [***KAFKA TRUSTSTORE PASSWORD KEY***]
              path: [***KAFKA TRUSTSTORE PASSWORD FILE***]
        - create: false
          secretRef: [***KAFKA SASL.JAAS.CONFIG SECRET***]
          items:
            - key: [***KAFKA SASL.JAAS.CONFIG KEY***]
              path: [***KAFKA SASL.JAAS.CONFIG FILE***]
      surveyorConfig:
        surveyor:
          authentication:
            ldap:
              enabled: true
        quarkus:
          security:
            ldap:
              dir-context:
                url: ldaps://openldap-chart.ldap:1390
                principal: ${LDAP_PRINCIPAL}
                password: ${LDAP_PASSWORD}
              identity-mapping:
                rdn-identifier: uid
                search-base-dn: ou=users,dc=openldap-chart,dc=ldap
                attribute-mappings:
                  "0":
                    from: cn
                    filter: (cn={0},ou=users,dc=openldap-chart,dc=ldap)
                    filter-base-dn: ou=users,dc=openldap-chart,dc=ldap
env:
```

```

- name: LDAP_PRINCIPAL
  valueFrom:
    secretKeyRef:
      name: [***LDAP CREDENTIALS SECRET***]
      key: principal
- name: LDAP_PASSWORD
  valueFrom:
    secretKeyRef:
      name: [***LDAP CREDENTIALS SECRET***]
      key: password
tlsConfigs:
  enabled: true
  secretRef: [***SURVEYOR TLS CERT SECRET***]
ingress:
  enabled: true
  className: nginx
  rules:
    path: "/"
    host: "MY-DOMAIN.EXAMPLE.COM"
  tls:
    enabled: true
    secretRef: "[***INGRESS TLS CERT SECRET***]"
globalTruststore:
  secretRef:
    name: [***LDAP TRUSTSTORE SECRET***]
    key: [***TRUSTSTORE SECRET KEY***]
  type: PKCS12
  password:
    secretRef:
      name: [***LDAP TRUSTSTORE SECRET***]
      key: [***TRUSTSTORE PW SECRET KEY***]

```

For clusterConfigs

clusterConfigs specifies the Kafka clusters that Cloudera Surveyor connects to. Clusters specified here are the ones that will be available on the UI for monitoring and management. For more information and additional examples, see [Registering Kafka clusters](#).

- 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.clusters[*].commonClientConfig – Kafka client configuration properties applied to all clients for this cluster. Must contain upstream Kafka client properties as a map. The exact properties that you specify here depend on the security configuration of the Kafka cluster that you want to connect. This example specifies a Kafka cluster that uses PLAIN authentication with TLS.

Sensitive property values are referenced from `Secrets` instead of being hardcoded. `Secrets` containing sensitive properties are mounted using the `secretsToMount` property. References use Kafka `DirectoryConfigProvider` syntax.



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

- clusterConfigs.clusters[*].authorization.enabled – Enables or disables authorization for this cluster.

For secretsToMount

`secretsToMount` specifies the `Secrets` to mount in the Cloudera Surveyor Container. You use `secretsToMount` to mount the sensitive values required for Kafka client configuration specified in `clusterConfigs`. For more information and additional examples, see [Managing sensitive data in client configuration](#)

- `secretsToMount[*].create` – Specifies whether to create the `Secret`. Set to `false` in this example as the `Secrets` are assumed to already exist.
- `secretsToMount[*].secretRef` – The name of the `Secret` to mount.
- `secretsToMount[*].items[*].key` – The key in the `Secret` to mount.
- `secretsToMount[*].items[*].path` – The path where the item is mounted. The path is relative to `/opt/secrets/[***SECRET NAME***]/` in the Cloudera Surveyor Container.

For `surveyorConfig`

`surveyorConfig` specifies global configuration for Cloudera Surveyor. This example sets various authentication properties.

- `surveyorConfig.surveyor.authentication ldap.enabled` - Enables or disables LDAP authentication. Set to `true` by default. Included in the example as a reference, you do not need to set the property explicitly to enable authentication.
- `surveyorConfig.quarkus.security.ldap.dir-context.*` - These properties configure the LDAP server that Cloudera Surveyor connects to. They specify the server URL, the distinguished name (DN) of the bind user, and the password of the bind user. These are required for establishing a secure connection with the LDAP directory.

The bind user credentials (principal and password) are referenced from the `LDAP_PRINCIPAL` and `LDAP_PASSWORD` environment variables. These environment variables are set using the `env` property. Their contents are referenced from a `Secret` that you created in a previous step.



Important: To ensure that Cloudera Surveyor connects to the LDAP server securely, the URL you specify in `surveyorConfig.quarkus.security.ldap.dir-context.url` must start with `ldaps://`.

- `surveyorConfig.quarkus.security.ldap.identity-mapping.*` - These properties configure how Cloudera Surveyor interacts with the LDAP directory to identify users and map their group memberships. They define the attributes and base DN's used to locate user entries and groups in the directory, as well as the filters applied to verify group membership.

For more information regarding `surveyorConfig.quarkus.*` properties, see [Using Security with an LDAP Realm](#) in the Quarkus documentation.



Note: Groups are not required for authentication. You might want to configure the `surveyorConfig.quarkus.security.ldap.identity-mapping.attribute-mappings.*` properties accordingly.

For `tlsConfigs`

`tlsConfigs` enables TLS and specifies the `Secret` containing the certificate of Cloudera Surveyor.

- `tlsConfigs.enabled` – Enables or disables TLS. Set to `true` by default. Included in the example as a reference, you do not need to set the property explicitly to enable TLS.
- `tlsConfigs.secretRef` – Name of the `Secret` containing the Cloudera Surveyor certificate and key.

For `ingress`

`ingress` enables the creation of an `Ingress`. The `Ingress` provides secure external access to the Cloudera Surveyor UI.

- `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 Ingress TLS certificates.



Note: If you are using `cert-manager`, add the `cert-manager.io/issuer: [**ISSUER NAME**]` annotation to the `ingress.extraAnnotations` property. If this annotation is set, a certificate is requested automatically and saved to the `Secret` specified in `ingress.tls.secretRef`.

For `globalTruststore`

`globalTruststore` specifies the `Secrets` containing the truststore of the LDAP server and the password for the truststore.

- `globalTruststore.secretRef.name` – The name of the Kubernetes `Secret` containing the truststore of the LDAP server.
- `globalTruststore.secretRef.key` – The key in the Kubernetes `Secret` that contains the truststore.
- `globalTruststore.password.name` – The name of the Kubernetes `Secret` containing the truststore password.
- `globalTruststore.password.key` – The key in the Kubernetes `Secret` that contains the truststore password.

9. Install Cloudera Surveyor with `helm` `install`.

```
helm install cloudera-surveyor \
  --namespace [***NAMESPACE***] \
  --values [***VALUES FILE***] \
  --set 'image.imagePullSecrets=[***REGISTRY CREDENTIALS SECRET***]' \
  --set-file clouderaLicense.fileContent=[***PATH TO LICENSE FILE***] \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surveyor \
  --version 1.7.0-b248
```

- The string `cloudera-surveyor` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- **[***VALUES FILE***]** is the values file you prepared in Step 8 on page 21.
- `imagePullSecrets` specifies what `Secret` is used to pull images from the Cloudera registry. Setting this property is mandatory, otherwise, Helm cannot pull the necessary images from the Cloudera Docker registry. Ensure that you replace **[***REGISTRY CREDENTIALS SECRET***]** with the name of the `Secret` you created in Step 3 on page 19.
- `clouderaLicense.fileContent` is used to register your license. If this property is set, a `Secret` is generated that contains the license you specify. Setting this property is mandatory. Cloudera Surveyor will not function without a valid license. Ensure that you replace **[***PATH TO LICENSE FILE***]** with the full path to your Cloudera license file.
- You can use `--set` to override properties that are defined in your values file, or add additional properties that are not present in your values file.

10. Verify your installation.

This is done by listing the `Deployments` and `Pods` in your namespace. If installation is successful, a Cloudera Surveyor `Deployment` and two `Pods` will be present in the cluster.

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

NAME		READY	UP-TO-DATE	AVAILABLE	AGE
#...					
cloudera-surveyor	2/2	2	2	13m	

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

NAME	READY	STATUS	RESTARTS	AGE
#...				

cloudera-surveyor-649f755f6d-689gx 13m	1/1	Running	0
cloudera-surveyor-649f755f6d-xj2kp 13m	1/1	Running	0

11. Access the Cloudera Surveyor UI.

The UI is accessible by connecting to the Ingress.

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

NAME PORTS	CLASS	HOSTS	ADDRESS
cloudera-surveyor-ingress 443	nginx	my-domain.example.com	10.14.91.1 80,

Typically you will be able to access the UI through the host and port listed. However, the exact port that you have to use might be infrastructure dependent. If you are unable to connect, check the configuration of your Ingress controller. You can also try connecting directly to the Service of the Ingress.

Results

Cloudera Surveyor is installed. You can now manage and monitor your Kafka clusters using the UI.

Related Information

[Registering Kafka clusters](#)

[Using Security with an LDAP Realm | Quarkus](#)

Installing Cloudera Surveyor in an air-gapped environment

Complete these steps to install Cloudera Surveyor if your Kubernetes cluster does not have internet access or if you want to install from a self-hosted registry. These steps install a fully secure instance of Cloudera Surveyor that has authentication, authorization, and channel encryption configured. The example configuration also demonstrates how you can connect a secure Kafka cluster to Cloudera Surveyor.

Before you begin

- General prerequisites:
 - Your Kubernetes environment meets requirements listed in [System requirements](#).
 - A self-hosted Docker registry is required. Your registry must be accessible by your Kubernetes cluster.
 - A machine with Internet connectivity is required. While the Kubernetes cluster does not need internet access, you will need a machine to pull the images from the Cloudera Docker registry.
 - Access to docker or equivalent utility that you can use to pull and push images is required. The following steps use docker. Replace commands where necessary.
 - You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
 - You have access to a valid Cloudera license.
 - Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- Prerequisites for channel encryption (TLS):
 - An Ingress controller is installed in your Kubernetes cluster. These steps use the [Ingress-Nginx controller](#).
 - Optional: cert-manager is installed in your Kubernetes cluster.
- Prerequisites for LDAP authentication:

- An LDAP server is available that has TLS enabled.
- The server is accessible from the Kubernetes cluster where Cloudera Surveyor is deployed.
- Entries containing usernames and passwords are located under a common base in the directory information tree. Passwords must be stored in the userPassword attribute in the user entries.
- If deploying Cloudera Surveyor with FIPS mode enabled (fipsMode: true), you must manually generate and configure an authentication key as part of installation.

The key must be generated and saved to a file, then configured using the --set-file option in your helm install command. For more information, see [Configuring LDAP authentication](#).

- Prerequisites for authorization:
 - Ensure that Kafka ACLs are set up for your Kafka cluster. Cloudera Surveyor uses Kafka ACLs to provide authorization.

Procedure

1. Copy the following installation artifacts to your self-hosted registry.

Table 8: Cloudera Surveyor artifacts on the Cloudera Docker registry

Artifact	Location	Description
Cloudera Surveyor server Docker image	container.repository.cloudera.com/cloudera/surveyor:0.2.0-csmop-1.7.0-b248	Docker image used for deploying Cloudera Surveyor.
Cloudera Surveyor UI application Docker image	container.repository.cloudera.com/cloudera/surveyor-app:0.2.0-csmop-1.7.0-b248	Docker image used for deploying Cloudera Surveyor.
Cloudera Surveyor Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surveyor:1.7.0-b248	Helm chart used to install Cloudera Surveyor with helm-install.



Note: The images are built for linux/arm64 and linux/amd64 architectures.

This step involves pulling the artifacts from the Cloudera Docker registry, retagging them, and then pushing them to your self-hosted registry. The exact steps you need to carry out depend on your environment and how your registry is set up. The following substeps demonstrate the basic workflow using docker and helm.



Tip: If your registry uses a non-default port, you might need to specify the registry port in helm as well as docker commands.

- a) Log in to the Cloudera Docker registry with both docker and helm.
Provide your Cloudera credentials when prompted.

```
docker login container.repository.cloudera.com
```

```
helm registry login container.repository.cloudera.com
```

- b) Log in to your self-hosted registry with both docker and helm.
Provide credentials when prompted.

```
docker login [***YOUR REGISTRY***]
```

```
helm registry login [***YOUR REGISTRY***]
```

- c) Pull the Docker images from the Cloudera Docker registry.

```
docker pull \
  --platform [***PLATFORM/ARCHITECTURE***] \
```

```
container.repository.cloudera.com/cloudera/[***IMAGE
NAME***]:[***VERSION***]
```

- d) Pull the Cloudera Surveyor Helm chart.

```
helm pull \
oci://container.repository.cloudera.com/cloudera-helm/csm-operator/sur
veyor \
--version 1.7.0-b248
```

- e) Retag the Docker images you pulled so that they contain the address of your registry.

```
docker tag \
[***ORIGINAL IMAGE TAG***] \
[***YOUR REGISTRY***]/cloudera/[***IMAGE NAME***]:[***VERSION***]
```

- f) Push the images and chart to your self-hosted registry.

```
docker push \
[***YOUR REGISTRY***]/cloudera/[***IMAGE NAME***]:[***VERSION***]
```

```
helm push \
surveyor-1.7.0-b248.tgz \
oci://[***YOUR REGISTRY***]/cloudera-helm/csm-operator
```

2. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Cloudera Surveyor. Use the namespace you create in all installation steps that follow.

3. Create a Kubernetes Secret containing credentials for your self-hosted registry.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
--namespace [***NAMESPACE***] \
--docker-server [***YOUR REGISTRY***] \
--docker-username [***USERNAME***] \
--docker-password "$(echo -n 'Enter your password: ' >&2; read -s passw
ord; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.
- Replace `[***USERNAME***]` with a username valid for your self-hosted registry.
- Enter the password for the user when prompted.

4. Create Secrets for sensitive Kafka client configuration values.

Cloudera Surveyor connects to Kafka clusters as any other Kafka client and requires a client configuration. If the Kafka cluster is secured, the client configuration will include sensitive property values. Cloudera recommends that you store sensitive values in Secrets, mount the Secrets to the Cloudera Surveyor Container, and reference the values in your configuration instead of hard-coding them.

Typically, you need to create two Secrets. One contains the Kafka cluster truststore and password, and the other contains a JAAS configuration. The Kafka truststore must be in JKS or PKCS12 format.

```
kubectl create secret generic [***KAFKA TRUSTSTORE SECRET***] \
--namespace [***NAMESPACE***] \
--from-file=[***KAFKA TRUSTSTORE KEY***]=[***PATH TO TRUSTSTORE***] \
```

```
--from-file=[***KAFKA TRUSTSTORE PASSWORD KEY***]=[***PATH TO TRUSTSTORE
PASSWORD FILE***]
```

```
kubectl create secret generic [***KAFKA SASL.JAAS.CONFIG SECRET***] \
--namespace [***NAMESPACE***] \
--from-file=[***KAFKA SASL.JAAS.CONFIG KEY***]=[***PATH TO KAFKA
SASL.JAAS.CONFIG FILE***]
```

- Take note of the `Secret` names as well as the key names you configure. You will need to specify them in a later step.
- All key names like `[***KAFKA TRUSTSTORE KEY***]` or `[***KAFKA SASL.JAAS.CONFIG KEY***]` are arbitrary.
- `[***PATH TO KAFKA SASL.JAAS.CONFIG FILE***]` is a path to a file containing JAAS configuration similar to the following example:

```
org.apache.kafka.common.security.plain.PlainLoginModule required username=
"MY-USER" password="MY-PASSWORD";
```

The contents of this file are set in a later step as the value of the `sasl.jaas.config` Kafka client property for Cloudera Surveyor internal Kafka clients. Ensure that the format of the configuration is valid for the `sasl.jaas.config` property. That is, its a single line of configuration.

5. Prepare a certificate and private key for Cloudera Surveyor.

- If you have `cert-manager` available, create a `Certificate` resource. Take note of the `Secret` name you configure in `spec.secretname` of the `Certificate` resource, you will need to specify it in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. Take note of the `Secret` name, you will need to specify it in a later step.

This `Secret` is referred to as `[***SURVEYOR TLS CERT SECRET***]` in the following steps.

6. Prepare a certificate and private key for Ingress.

- If you have `cert-manager` available, the certificate and private key for `Ingress` are automatically requested by the `Ingress`. You only need to ensure that you have a valid `Issuer` available in `cert-manager`. You specify the name of the `Issuer` resource in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. Take note of the `Secret` name, you will need to specify it in a later step.

This `Secret` referred to as `[***INGRESS TLS CERT SECRET***]` in the following steps.

7. Set up resources for LDAP authentication.

- Generate a Java truststore (PKCS12 or JKS) containing the TLS certificate of the root Certificate Authority (CA) of the LDAP certificate chain.

```
keytool -import -trustcacerts -file [***LDAP ROOT CA***] \
-keystore [***TRUSTSTORE NAME***] \
-storepass [***TRUSTSTORE PASSWORD***] \
-storetype PKCS12
```

- Create a `Secret` containing the truststore and the truststore password.

```
kubectl create secret generic [***LDAP TRUSTSTORE SECRET***] \
--namespace [***NAMESPACE***] \
--from-file=[***TRUSTSTORE SECRET KEY***]=[***TRUSTSTORE NAME***] \
--from-file=[***TRUSTSTORE PW SECRET KEY***]=[***PATH TO TRUSTSTORE PW
FILE***]
```

Take note of `[***LDAP TRUSTSTORE SECRET***]`, `[***TRUSTSTORE SECRET KEY***]`, and `[***TRUSTSTORE PW SECRET KEY***]`. You will need to specify these names in a custom values file you create in a later step.

- c) Create a Secret containing your LDAP principal and password (bind credentials).

```
kubectl create secret generic [***LDAP CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-literal=principal="$(echo -n 'Enter principal: ' >&2; read -s
principal; echo >&2; echo $principal)" --from-literal=password="$(echo
-n 'Enter password: ' >&2; read -s password; echo >&2; echo $password)"
```

- Take note of [***LDAP CREDENTIALS SECRET***]. You will need to specify this name in a custom values file you create in a later step.
- Enter your principal and the password for the principal when prompted. Example principal:

```
cn=admin,dc=openldap-chart,dc=ldap
```

8. Prepare a custom values file (values.yaml).

The values file contains configuration for Cloudera Surveyor. This file specifies the Kafka clusters that Cloudera Surveyor connects to as well as various other configuration properties.

```
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/[***KAFKA TRUSTSTORE
SECRET***]/[***KAFKA TRUSTSTORE FILE***]"
        ssl.truststore.password: "\\${dir:/opt/secrets/[***KAFKA
TRUSTSTORE SECRET***]:[***KAFKA TRUSTSTORE PASSWORD FILE***]}"
        sasl.jaas.config: "\\${dir:/opt/secrets/[***KAFKA SASL.JAAS.CONFIG
SECRET***]:[***KAFKA SASL.JAAS.CONFIG FILE***]}"
        adminOperationTimeout: PT1M
        authorization:
          enabled: true
      secretsToMount:
        - create: false
          secretRef: [***KAFKA TRUSTSTORE SECRET***]
          items:
            - key: [***KAFKA TRUSTSTORE KEY***]
              path: [***KAFKA TRUSTSTORE FILE***]
            - key: [***KAFKA TRUSTSTORE PASSWORD KEY***]
              path: [***KAFKA TRUSTSTORE PASSWORD FILE***]
        - create: false
          secretRef: [***KAFKA SASL.JAAS.CONFIG SECRET***]
          items:
            - key: [***KAFKA SASL.JAAS.CONFIG KEY***]
              path: [***KAFKA SASL.JAAS.CONFIG FILE***]
      surveyorConfig:
        surveyor:
          authentication:
            ldap:
              enabled: true
        quarkus:
          security:
            ldap:
              dir-context:
                url: ldaps://openldap-chart.ldap:1390
```

```

    principal: ${LDAP_PRINCIPAL}
    password: ${LDAP_PASSWORD}
  identity-mapping:
    rdn-identifier: uid
    search-base-dn: ou=users,dc=openldap-chart,dc=ldap
    attribute-mappings:
      "0":
        from: cn
        filter: (cn={0},ou=users,dc=openldap-chart,dc=ldap)
        filter-base-dn: ou=users,dc=openldap-chart,dc=ldap
env:
  - name: LDAP_PRINCIPAL
    valueFrom:
      secretKeyRef:
        name: [***LDAP CREDENTIALS SECRET***]
        key: principal
  - name: LDAP_PASSWORD
    valueFrom:
      secretKeyRef:
        name: [***LDAP CREDENTIALS SECRET***]
        key: password
tlsConfigs:
  enabled: true
  secretRef: [***SURVEYOR TLS CERT SECRET***]
ingress:
  enabled: true
  className: nginx
  rules:
    path: "/"
    host: "MY-DOMAIN.EXAMPLE.COM"
  tls:
    enabled: true
    secretRef: "[***INGRESS TLS CERT SECRET***]"
globalTruststore:
  secretRef:
    name: [***LDAP TRUSTSTORE SECRET***]
    key: [***TRUSTSTORE SECRET KEY***]
  type: PKCS12
  password:
    secretRef:
      name: [***LDAP TRUSTSTORE SECRET***]
      key: [***TRUSTSTORE PW SECRET KEY***]

```

For clusterConfigs

clusterConfigs specifies the Kafka clusters that Cloudera Surveyor connects to. Clusters specified here are the ones that will be available on the UI for monitoring and management. For more information and additional examples, see [Registering Kafka clusters](#).

- 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.clusters[*].commonClientConfig – Kafka client configuration properties applied to all clients for this cluster. Must contain upstream Kafka client properties as a map. The exact properties that you specify here depend on the security configuration of the Kafka cluster that you want to connect. This example specifies a Kafka cluster that uses PLAIN authentication with TLS.

Sensitive property values are referenced from `Secrets` instead of being hardcoded. `Secrets` containing sensitive properties are mounted using the `secretsToMount` property. References use Kafka `DirectoryConfigProvider` syntax.



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

- `clusterConfigs.clusters[*].authorization.enabled` – Enables or disables authorization for this cluster.

For `secretsToMount`

`secretsToMount` specifies the `Secrets` to mount in the Cloudera Surveyor Container. You use `secretsToMount` to mount the sensitive values required for Kafka client configuration specified in `clusterConfigs`. For more information and additional examples, see [Managing sensitive data in client configuration](#)

- `secretsToMount[*].create` – Specifies whether to create the `Secret`. Set to `false` in this example as the `Secrets` are assumed to already exist.
- `secretsToMount[*].secretRef` – The name of the `Secret` to mount.
- `secretsToMount[*].items[*].key` – The key in the `Secret` to mount.
- `secretsToMount[*].items[*].path` – The path where the item is mounted. The path is relative to `/opt/secrets/[***SECRET NAME***]/` in the Cloudera Surveyor Container.

For `surveyorConfig`

`surveyorConfig` specifies global configuration for Cloudera Surveyor. This example sets various authentication properties.

- `surveyorConfig.surveyor.authentication.ldap.enabled` - Enables or disables LDAP authentication. Set to `true` by default. Included in the example as a reference, you do not need to set the property explicitly to enable authentication.
- `surveyorConfig.quarkus.security.ldap.dir-context.*` - These properties configure the LDAP server that Cloudera Surveyor connects to. They specify the server URL, the distinguished name (DN) of the bind user, and the password of the bind user. These are required for establishing a secure connection with the LDAP directory.

The bind user credentials (principal and password) are referenced from the `LDAP_PRINCIPAL` and `LDAP_PASSWORD` environment variables. These environment variables are set using the `env` property. Their contents are referenced from a `Secret` that you created in a previous step.



Important: To ensure that Cloudera Surveyor connects to the LDAP server securely, the URL you specify in `surveyorConfig.quarkus.security.ldap.dir-context.url` must start with `ldaps://`.

- `surveyorConfig.quarkus.security.ldap.identity-mapping.*` - These properties configure how Cloudera Surveyor interacts with the LDAP directory to identify users and map their group memberships. They define the attributes and base DN's used to locate user entries and groups in the directory, as well as the filters applied to verify group membership.

For more information regarding `surveyorConfig.quarkus.*` properties, see [Using Security with an LDAP Realm](#) in the Quarkus documentation.



Note: Groups are not required for authentication. You might want to configure the `surveyorConfig.quarkus.security.ldap.identity-mapping.attribute-mappings.*` properties accordingly.

For `tlsConfigs`

`tlsConfigs` enables TLS and specifies the `Secret` containing the certificate of Cloudera Surveyor.

- `tlsConfigs.enabled` – Enables or disables TLS. Set to `true` by default. Included in the example as a reference, you do not need to set the property explicitly to enable TLS.
- `tlsConfigs.secretRef` – Name of the `Secret` containing the Cloudera Surveyor certificate and key.

For `ingress`

ingress enables the creation of an Ingress. The Ingress provides secure external access to the Cloudera Surveyor UI.

- `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 Ingress TLS certificates.



Note: If you are using cert-manager, add the `cert-manager.io/issuer: [**ISSUER NAME**]` annotation to the `ingress.extraAnnotations` property. If this annotation is set, a certificate is requested automatically and saved to the Secret specified in `ingress.tls.secretRef`.

For globalTruststore

`globalTruststore` specifies the Secrets containing the truststore of the LDAP server and the password for the truststore.

- `globalTruststore.secretRef.name` – The name of the Kubernetes Secret containing the truststore of the LDAP server.
- `globalTruststore.secretRef.key` – The key in the Kubernetes Secret that contains the truststore.
- `globalTruststore.password.name` – The name of the Kubernetes Secret containing the truststore password.
- `globalTruststore.password.key` – The key in the Kubernetes Secret that contains the truststore password.

9. Install Cloudera Surveyor with helm install.

```
helm install cloudera-surveyor \
  --namespace [***NAMESPACE***] \
  --values [***VALUES FILE***] \
  --set image.registry=[***YOUR REGISTRY***] \
  --set 'image.imagePullSecrets=[***REGISTRY CREDENTIALS SECRET***]' \
  --set-file clouderaLicense.fileContent=[***PATH TO LICENSE FILE***] \
  oci://[***YOUR REGISTRY***]/cloudera-helm/csm-operator/surveyor \
  --version 1.7.0-b248
```

- The string `cloudera-surveyor` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- `[***VALUES FILE***]` is the values file you prepared in Step 8 on page 29.
- `imagePullSecrets` specifies what Secret is used to pull images from the specified registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the Secret you created in Step 3 on page 27.
- `clouderaLicense.fileContent` is used to register your license. If this property is set, a Secret is generated that contains the license you specify. Setting this property is mandatory. Cloudera Surveyor will not function without a valid license. Ensure that you replace `[***PATH TO LICENSE FILE***]` with the full path to your Cloudera license file.
- You can use `--set` to override properties that are defined in your values file, or add additional properties that are not present in your values file.

10. Verify your installation.

This is done by listing the Deployments and Pods in your namespace. If installation is successful, a Cloudera Surveyor Deployment and two Pods will be present in the cluster.

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

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
------	-------	------------	-----------	-----

#...	cloudera-surveyor	2/2	2	2	13m
------	-------------------	-----	---	---	-----


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


NAME	READY	STATUS	RESTARTS	AGE
#...				
cloudera-surveyor-649f755f6d-689gx	1/1	Running		0
13m				
cloudera-surveyor-649f755f6d-xj2kp	1/1	Running		0
13m				

11. Access the Cloudera Surveyor UI.

The UI is accessible by connecting to the Ingress.

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


NAME	CLASS	HOSTS	ADDRESS
PORTS			
cloudera-surveyor-ingress	nginx	my-domain.example.com	10.14.91.1 80, 443

Typically you will be able to access the UI through the host and port listed. However, the exact port that you have to use might be infrastructure dependent. If you are unable to connect, check the configuration of your Ingress controller. You can also try connecting directly to the Service of the Ingress.

Results

Cloudera Surveyor is installed. You can now manage and monitor your Kafka clusters using the UI.

Related Information

[Registering Kafka clusters](#)

[Using Security with an LDAP Realm | Quarkus](#)

Installing Cloudera Surveyor for evaluation

Complete these steps to install a basic deployment of Cloudera Surveyor that has no security configured. Use these instructions if you want to install quickly in a development environment for proof of concept or evaluation purposes.

Before you begin

- Your Kubernetes environment meets requirements listed in [System requirements](#).
- Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Cloudera Surveyor. Use the namespace you create in all installation steps that follow.

2. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

3. Create a Kubernetes Secret containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
  -s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.
 - Replace `[***USERNAME***]` with your Cloudera username.
 - Enter your Cloudera password when prompted.
4. Prepare a custom values file (values.yaml).

The values file contains configuration for Cloudera Surveyor. This file specifies the Kafka clusters that Cloudera Surveyor connects to as well as various other configuration properties.

```
service:
  type: NodePort
clusterConfigs:
  clusters:
    - clusterName: [***CLUSTER NAME***]
      tags:
        - [***TAG1***]
        - [***TAG2***]
      bootstrapServers: [***BOOTSTRAP SERVERS***]
      adminOperationTimeout: PT1M
      authorization:
        enabled: false
      commonClientConfig:
        security.protocol: PLAINTEXT
surveyorConfig:
  surveyor:
    authentication:
      ldap:
        enabled: false
  tlsConfigs:
    enabled: false
```

- `service.type` – The type of Kubernetes Service created for Cloudera Surveyor. Set to `NodePort` here to make the UI reachable from outside the Kubernetes cluster without configuring an Ingress or LoadBalancer.
- `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[*].commonClientConfig` – Kafka client configuration properties applied to all clients for this cluster. Must contain upstream Kafka client properties as a map. The exact properties that you specify here depend on the security configuration of the Kafka cluster that you want to connect. This example specifies a Kafka cluster that is unsecure. For more information, see *Registering Kafka clusters*.
- All security-related properties are set `false` to disable security. These properties must be explicitly set to `false` as the default value for all of them is `true`.

5. Install Cloudera Surveyor with helm `install`.

```
helm install cloudera-surveyor \
  --namespace [***NAMESPACE***] \
  --values [***VALUES FILE***] \
  --set 'image.imagePullSecrets=[***REGISTRY CREDENTIALS SECRET***]' \
  --set-file clouderaLicense.fileContent=[***PATH TO LICENSE FILE***] \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surveyor \
  --version 1.7.0-b248
```

- The string `cloudera-surveyor` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- `[***VALUES FILE***]` is the values file you prepared in Step 4 on page 34.
- `imagePullSecrets` specifies what secret is used to pull images from the Cloudera registry. Setting this property is mandatory, otherwise, Helm cannot pull the necessary images from the Cloudera Docker registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the `Secret` you created in Step 3 on page 34.
- `clouderaLicense.fileContent` is used to register your license. If this property is set, a `Secret` is generated that contains the license you specify. Setting this property is mandatory. Cloudera Surveyor will not function without a valid license. Ensure that you replace `[***PATH TO LICENSE FILE***]` with the full path to your Cloudera license file.
- You can use `--set` to override properties that are defined in your values file, or add additional properties that are not present in your values file.

6. Verify your installation.

This is done by listing the `Deployments` and `Pods` in your namespace. If installation is successful, a Cloudera Surveyor `Deployment` and two `Pods` will be present in the cluster.

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

NAME		READY	UP-TO-DATE	AVAILABLE	AGE
#...					
cloudera-surveyor	2/2	2	2	13m	

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

NAME	READY	STATUS	RESTARTS	AGE
#...				
cloudera-surveyor-649f755f6d-689gx	1/1	Running		0
13m				
cloudera-surveyor-649f755f6d-xj2kp	1/1	Running		0
13m				

7. Access the Cloudera Surveyor UI.

The NodePort type Service configured in the values file makes the UI accessible from any of the Kubernetes cluster nodes on the external port of the Service. List Services to get the external port.

```
kubectl get service cloudera-surveyor-service --namespac
e [***NAMESPACE***]
```

NAME (S)	AGE	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT
cloudera-surveyor-service	0:30525/TCP	NodePort	10.43.196.52	<none>	808

In this example, the external port is 30525.

Results

Cloudera Surveyor is installed. You can now manage and monitor your Kafka clusters using the UI.

Related Information

[Registering Kafka clusters](#)

Installing Schema Registry with Helm

Learn how to install Schema Registry in Cloudera Streams Messaging Operator for Kubernetes with Helm. Schema Registry is a standalone application that allows you to efficiently store and manage schemas for your streaming data.

Schema Registry is installed in your Kubernetes cluster with the Schema Registry Helm chart using the `helm install` command. When you install the chart, Helm deploys an instance of Schema Registry, which provides you with schema storage and management capabilities.

During installation, you configure Schema Registry using a custom values file (`values.yaml`) passed to the Helm chart with the `--values (-f)` option. This file contains properties for configuring Schema Registry, including network access, database connectivity, and security settings for TLS and OAuth authentication. Additionally, some properties are configured with `--set` options.

Installation instructions are provided for the following scenarios.

- **Installing in an internet environment** – Follow these steps to install a fully secure instance of Schema Registry in a Kubernetes cluster with internet access.
- **Installing in an air-gapped environment** – Follow these steps to install a fully secure instance of Schema Registry in a Kubernetes cluster without internet access or if you want to install from a self-hosted registry.
- **Installing for evaluation** – Follow these steps to install an unsecure instance of Schema Registry for development or proof of concept purposes.

Installing Schema Registry in an internet environment

Complete these steps to install Schema Registry if your Kubernetes cluster has internet access. These steps install a fully secure instance of Schema Registry that has authentication, authorization, and channel encryption configured, leveraging a PostgreSQL database for persistent schema storage.

Before you begin

- General prerequisites:
 - Your Kubernetes environment meets requirements listed in [System requirements](#).
 - Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.

- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- Prerequisites for channel encryption (TLS):
 - An Ingress controller is installed in your Kubernetes cluster. These steps use the [Ingress-NGinx controller](#).
 - Optional: cert-manager is installed in your Kubernetes cluster.
- Prerequisites for OAuth authentication:
 - An OAuth server is available that has TLS enabled.
 - The server is accessible from the Kubernetes cluster where Schema Registry is deployed.
 - At least one client must be configured in your realm that supports Client Credentials flow (sometimes referred to as Machine-to-Machine (M2M), Service Account, or Application Permissions).
 - Identify if your OAuth server issues tokens that contain a value in the aud claim. If a value is present, note it down as you will need to provide it in your configuration. Referred to as `***OAUTH EXPECTED AUDIENCE***` in the following steps.
 - Get the JWKS endpoint URL of your OAuth server. You will need to provide it in your configuration. Schema Registry requires this endpoint to validate the signatures of incoming tokens. Referred to as `***OAUTH JWKS URL***` in the following steps.
 - Identify which JWT claim in your token contains the username to authorize. Schema Registry checks the sub claim by default. If your provider uses a different field, note it down as you will need to provide it in your configuration. Referred to as `***OAUTH PRINCIPAL CLAIM***` in the following steps.
 - Collect the usernames that you want to set as admin and read-only users. You will provide these in your configuration. Referred to as `***ADMIN USERS***` and `***READ-ONLY USERS***` in the following steps.
- Database prerequisites for persistent storage:
 - You have access to a PostgreSQL server with TLS and have provisioned a database for Schema Registry.
 - Get the JDBC URL for the PostgreSQL server. Referred to as `***POSTGRES SQL JDBC URL***` in the following steps.
 - Get a username that Schema Registry can use to connect to the PostgreSQL server. Referred to as `***POSTGRES SQL USERNAME***`.

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Schema Registry. Use the namespace you create in all installation steps that follow.

2. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

3. Create a Kubernetes Secret containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
```

```
--docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
-s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.
- Replace `[***USERNAME***]` with your Cloudera username.
- Enter your Cloudera password when prompted.

4. Prepare a keystore for Schema Registry.

- If you have cert-manager available, create a `Certificate` resource. Take note of the `Secret` name you configure in `spec.secretName` of the `Certificate` resource, you will need to specify it in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. The keystore should be in PKCS12 format.

```
kubectl create secret generic [***KEYSTORE SECRET NAME***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***KEYSTORE SECRET KEY***]=[***PATH TO KEYSTORE.P12***] \
  --from-file=[***KEYSTORE PASSWORD SECRET KEY***]=[***PATH TO KEYSTORE
  PASSWORD FILE***]
```

Take note of the `Secret` name, you will need to specify it in a later step.

5. Prepare a certificate and private key for Ingress.

- If you have cert-manager available, the certificate and private key for Ingress are automatically requested by the Ingress. You only need to ensure that you have a valid `Issuer` available in cert-manager. You specify the name of the `Issuer` resource in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. Take note of the `Secret` name, you will need to specify it in a later step.

This `Secret` referred to as `[***INGRESS TLS CERT SECRET***]` in the following steps.

6. Set up resources for OAuth authentication and authorization.

- Generate a Java truststore (PKCS12) containing the TLS certificate of the root Certificate Authority (CA) of the OAuth certificate chain.

```
keytool -import -trustcacerts -file [***OAUTH ROOT CA***] \
  -keystore [***TRUSTSTORE NAME***] \
  -storepass [***TRUSTSTORE PASSWORD***] \
  -storetype PKCS12
```

- Create a `Secret` containing the truststore and its password.

```
kubectl create secret generic [***OAUTH TRUSTSTORE SECRET NAME***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***OAUTH TRUSTSTORE SECRET KEY***]=[***TRUSTSTORE
  NAME***] \
  --from-file=[***OAUTH TRUSTSTORE PASSWORD SECRET KEY***]=[***PATH TO
  TRUSTSTORE PW FILE***]
```

Take note of `[***OAUTH TRUSTSTORE SECRET NAME***]`, `[***OAUTH TRUSTSTORE SECRET KEY***]`, and `[***OAUTH TRUSTSTORE PASSWORD SECRET KEY***]`.

7. Prepare Secrets for required PostgreSQL connection values.

Typically, you will need a `Secret` containing the PostgreSQL database password. Depending on your security requirements, an additional `Secret` might be required to hold TLS-related files, such as a Certificate Authority (CA) certificate used to establish a secure database connection.

- Create a `Secret` containing the PostgreSQL server password.

```
kubectl create secret generic [***POSTGRESQL PASSWORD SECRET NAME***] \
  --namespace [***NAMESPACE***] \
```

```
--from-file=[***POSTGRESQL PASSWORD SECRET KEY***]=[***PATH TO
DATABASE PASSWORD FILE***]
```

- b) Create a Secret containing TLS configuration files.

All files included in this Secret will be mounted to a common directory in the Pod, allowing the files to be referenced in your JDBC URL. In this example, a single CA certificate is added.

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

8. Prepare a custom values file (values.yaml).

The following example configures a fully secure deployment with a PostgreSQL database for persistent schema storage.

```
tls:
  enabled: true
  keystore:
    secretKeyRef:
      name: [***KEYSTORE SECRET NAME***]
      key: [***KEYSTORE SECRET KEY***]
    password:
      secretKeyRef:
        name: [***KEYSTORE SECRET NAME***]
        key: [***KEYSTORE PASSWORD SECRET KEY***]
    type: PKCS12

ingress:
  enabled: true
  className: "nginx"
  rules:
    path: "/"
    host: "my-domain.example.com"
  tls:
    enabled: true
    secretRef: [***INGRESS TLS CERT SECRET***]
  extraAnnotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

authentication:
  oauth:
    enabled: true
    jwt:
      principalClaimName: [***OAUTH PRINCIPAL CLAIM***]
      expectedAudience: [***OAUTH EXPECTED AUDIENCE***]
    jwks:
      url: [***OAUTH JWKS URL***]
    tls:
      truststore:
        secretKeyRef:
          name: [***OAUTH TRUSTSTORE SECRET NAME***]
          key: [***OAUTH TRUSTSTORE SECRET KEY***]
        password:
          secretKeyRef:
            name: [***OAUTH TRUSTSTORE SECRET NAME***]
            key: [***OAUTH TRUSTSTORE PASSWORD SECRET KEY***]
        type: PKCS12
  authorization:
    simple:
      enabled: true
      adminUsers: [***ADMIN USERS***]
      readOnlyUsers: [***READ-ONLY USERS***]
```

```

database:
  type: postgresql
  jdbcUrl: [***POSTGRESQL JDBC URL***]
  username: [***POSTGRESQL USERNAME***]
  password:
    secretKeyRef:
      name: [***POSTGRESQL PASSWORD SECRET NAME***]
      key: [***POSTGRESQL PASSWORD SECRET KEY***]
  tls:
    secretRef: [***POSTGRESQL TRUSTSTORE SECRET NAME***]

```

For tls

- `tls.enabled` – Enables or disables TLS.
- `tls.keystore.secretKeyRef.name` – The name of the `Secret` containing the TLS keystore.
- `tls.keystore.secretKeyRef.key` – The key in the `Secret` specified by `tls.keystore.secretKeyRef.name` that contains the TLS keystore.
- `tls.keystore.password.secretKeyRef.name` – The name of the `Secret` containing the TLS keystore password.
- `tls.keystore.password.secretKeyRef.key` – The key in the `Secret` specified by `tls.keystore.password.secretKeyRef.name` that contains the TLS keystore password.

For ingress

- `ingress.enabled` – Enables or disables external access through Ingress.
- `ingress.tls.enabled` – Enables or disables TLS for Ingress.
- `ingress.tls.secretRef` – The name of the `Secret` containing Ingress TLS certificates.
- `ingress.extraAnnotations.*` – Extra annotations to apply to the Ingress.



Note: If you are using `cert-manager`, add the `cert-manager.io/issuer: [**ISSUER NAME**]` annotation to the `ingress.extraAnnotations` property. If this annotation is set, a certificate is requested automatically and saved to the `Secret` specified in `ingress.tls.secretRef`.

For authentication

- `authentication.oauth.enabled` – Enables OAuth authentication for the Schema Registry server.
- `authentication.oauth.jwt.principalClaimName` – The name of the claim in the JWT token that contains the principal (username) used for authorization.
- `authentication.oauth.jwt.expectedAudience` – The expected audience value. If the JWT token contains an `aud` claim, it must match this value, otherwise the token is considered invalid.
- `authentication.oauth.jwks.url` – The URL to the JWKS endpoint.
- `authentication.oauth.jwks.tls.truststore.secretKeyRef.name` – The name of the `Secret` that contains the truststore for accessing the JWKS endpoint. Configure this property if the backend of your JWKS has self-signed certificates.
- `authentication.oauth.jwks.tls.truststore.secretKeyRef.key` – The key in the `Secret` specified by `authentication.oauth.jwks.tls.truststore.secretKeyRef.name` that contains the truststore for accessing the JWKS endpoint.
- `authentication.oauth.jwks.tls.truststore.password.secretKeyRef.name` – The name of the `Secret` that contains the truststore password for accessing the JWKS endpoint.
- `authentication.oauth.jwks.tls.truststore.password.secretKeyRef.key` – The key in the `Secret` specified by `authentication.oauth.jwks.tls.truststore.password.secretKeyRef.name` that contains the truststore password for accessing the JWKS endpoint.

For authorization

- `authorization.simple.enabled` – Enables or disables authorization.

- `authorization.simple.adminUsers` – A list of admin usernames. Admin users can perform any operation in Schema Registry.
- `authorization.simple.readOnlyUsers` – A list of read-only usernames. Read-only users can only perform read operations in Schema Registry.

For database

- `database.jdbcUrl` – The JDBC URL that points to your PostgreSQL database.
- `database.username` – The PostgreSQL username for Schema Registry database connections.
- `database.password.secretKeyRef.name` – The name of the `Secret` containing the PostgreSQL database password.
- `database.password.secretKeyRef.key` – The key in the `Secret` specified by `database.password.secretKeyRef.name` that contains the PostgreSQL database password.
- `database.tls.secretRef` – The name of a `Secret` containing TLS configuration for PostgreSQL connections (certificates, truststores, and so on). All keys from the `Secret` are mounted to `/etc/schema-registry/postgres/tls`. Reference mounted files in your JDBC URL (`database.jdbcUrl`) to configure SSL connections if SSL is required for PostgreSQL.

9. Install Schema Registry with helm install.

```
helm install schema-registry \
  --namespace [***NAMESPACE***] \
  --values [***VALUES FILE***] \
  --set 'image.imagePullSecrets=[***REGISTRY CREDENTIALS SECRET***]' \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/schema-registry \
  --version 1.7.0-b248
```

- The string `schema-registry` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- `[***VALUES FILE***]` is the values file you prepared in Step 8 on page 39.
- `imagePullSecrets` specifies what `Secret` is used to pull images from the Cloudera registry. Setting this property is mandatory, otherwise, Helm cannot pull the necessary images from the Cloudera Docker registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the `Secret` you created in Step 3 on page 37.
- You can use `--set` to override properties that are defined in your values file, or add additional properties that are not present in your values file.

10. Verify your installation.

This is done by listing the `Deployments` and `Pods` in your namespace. If installation is successful, a Schema Registry `Deployment` and two `Pods` will be present in the cluster.

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

NAME		READY	UP-TO-DATE	AVAILABLE	AGE
#...					
schema-registry	2/2	2	2	13m	

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

NAME	READY	STATUS	RESTARTS	AGE
#...				
schema-registry-858f647cfc-82mkj	1/1	Running		0
13m				
schema-registry-858f647cfc-jl4nt	1/1	Running		0
13m				

What to do next

Configure clients to interact with Schema Registry or review and use the REST API.

Related Information

[Schema Registry REST API reference](#)

Installing Schema Registry in an air-gapped environment

Complete these steps to install Schema Registry if your Kubernetes cluster does not have internet access or if you want to install from a self-hosted registry. These steps install a fully secure instance of Schema Registry that has authentication, authorization, and channel encryption configured, leveraging a PostgreSQL database for persistent schema storage.

Before you begin

- General prerequisites:
 - Your Kubernetes environment meets requirements listed in [System requirements](#).
 - A self-hosted Docker registry is required. Your registry must be accessible by your Kubernetes cluster.
 - A machine with Internet connectivity is required. While the Kubernetes cluster does not need internet access, you will need a machine to pull the images from the Cloudera Docker registry.
 - Access to docker or equivalent utility that you can use to pull and push images is required. The following steps use docker. Replace commands where necessary.
 - You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
 - Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- Prerequisites for channel encryption (TLS):
 - An Ingress controller is installed in your Kubernetes cluster. These steps use the [Ingress-Nginx controller](#).
 - Optional: cert-manager is installed in your Kubernetes cluster.
- Prerequisites for OAuth authentication:
 - An OAuth server is available that has TLS enabled.
 - The server is accessible from the Kubernetes cluster where Schema Registry is deployed.
 - At least one client must be configured in your realm that supports Client Credentials flow (sometimes referred to as Machine-to-Machine (M2M), Service Account, or Application Permissions).
 - Identify if your OAuth server issues tokens that contain a value in the aud claim. If a value is present, note it down as you will need to provide it in your configuration. Referred to as `***OAUTH EXPECTED AUDIENCE***` in the following steps.
 - Get the JWKS endpoint URL of your OAuth server. You will need to provide it in your configuration. Schema Registry requires this endpoint to validate the signatures of incoming tokens. Referred to as `***OAUTH JWKS URL***` in the following steps.
 - Identify which JWT claim in your token contains the username to authorize. Schema Registry checks the sub claim by default. If your provider uses a different field, note it down as you will need to provide it in your configuration. Referred to as `***OAUTH PRINCIPAL CLAIM***` in the following steps.
 - Collect the usernames that you want to set as admin and read-only users. You will provide these in your configuration. Referred to as `***ADMIN USERS***` and `***READ-ONLY USERS***` in the following steps.
- Database prerequisites for persistent storage:
 - You have access to a PostgreSQL server with TLS and have provisioned a database for Schema Registry.
 - Get the JDBC URL for the PostgreSQL server. Referred to as `***POSTGRES JDBC URL***` in the following steps.

- Get a username that Schema Registry can use to connect to the PostgreSQL server. Referred to as `[***POSTGRES SQL USERNAME***]`.

Procedure

1. Copy the following installation artifacts to your self-hosted registry.

Table 9: Schema Registry artifacts on the Cloudera Docker registry

Artifact	Location	Description
Schema Registry Docker image	container.repository.cloudera.com/cloudera/schema-registry:0.10.0-csmop-1.7.0-b248	Docker image used for deploying Schema Registry.
Schema Registry Helm chart	oci://container.repository.cloudera.com/cloudera-helm/csm-operator/schema-registry:1.7.0-b248	Helm chart used to install Schema Registry with helm-install.



Note: The images are built for linux/arm64 and linux/amd64 architectures.

This step involves pulling the artifacts from the Cloudera Docker registry, retagging them, and then pushing them to your self-hosted registry. The exact steps you need to carry out depend on your environment and how your registry is set up. The following substeps demonstrate the basic workflow using docker and helm.



Tip: If your registry uses a non-default port, you might need to specify the registry port in helm as well as docker commands.

- a) Log in to the Cloudera Docker registry with both docker and helm.

Provide your Cloudera credentials when prompted.

```
docker login container.repository.cloudera.com
```

```
helm registry login container.repository.cloudera.com
```

- b) Log in to your self-hosted registry with both docker and helm.

Provide credentials when prompted.

```
docker login [***YOUR REGISTRY***]
```

```
helm registry login [***YOUR REGISTRY***]
```

- c) Pull the Schema Registry Docker image from the Cloudera Docker registry.

```
docker pull \
  --platform [***PLATFORM/ARCHITECTURE***] \
  container.repository.cloudera.com/cloudera/schema-registry:0.10.0-
  csmop-1.7.0-b248
```

- d) Pull the Schema Registry Helm chart.

```
helm pull \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/sch
  ema-registry \
  --version 1.7.0-b248
```

- e) Retag the Schema Registry Docker image you pulled so that they contain the address of your registry.

```
docker tag \
```

```
container.repository.cloudera.com/cloudera/schema-registry:0.10.0-
csmop-1.7.0-b248 \
[***YOUR_REGISTRY***]/cloudera/[***IMAGE_NAME***]:[***VERSION***]
```

- f) Push the Schema Registry image and chart to your self-hosted registry.

```
docker push \
[***YOUR_REGISTRY***]/cloudera/[***IMAGE_NAME***]:[***VERSION***]
```

```
helm push \
schema-registry-1.7.0-b248.tgz \
oci://[***YOUR_REGISTRY***]/cloudera-helm/csm-operator
```

2. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

This is the namespace where you install Schema Registry. Use the namespace you create in all installation steps that follow.

3. Create a Kubernetes Secret containing credentials for your self-hosted registry.

```
kubectl create secret docker-registry [***REGISTRY_CREDENTIALS_SECRET***] \
--namespace [***NAMESPACE***] \
--docker-server [***YOUR_REGISTRY***] \
--docker-username [***USERNAME***] \
--docker-password "$(echo -n 'Enter your password: ' >&2; read -s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY_CREDENTIALS_SECRET***]`. You will need to specify the name in a later step.
- Replace `[***USERNAME***]` with a username valid for your self-hosted registry.
- Enter the password for the user when prompted.

4. Prepare a keystore for Schema Registry.

- If you have cert-manager available, create a `Certificate` resource. Take note of the `Secret` name you configure in `spec.secretName` of the `Certificate` resource, you will need to specify it in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. The keystore should be in PKCS12 format.

```
kubectl create secret generic [***KEYSTORE_SECRET_NAME***] \
--namespace [***NAMESPACE***] \
--from-file=[***KEYSTORE_SECRET_KEY***]=[***PATH TO KEYSTORE.P12***] \
--from-file=[***KEYSTORE_PASSWORD_SECRET_KEY***]=[***PATH TO KEYSTORE
PASSWORD FILE***]
```

Take note of the `Secret` name, you will need to specify it in a later step.

5. Prepare a certificate and private key for Ingress.

- If you have cert-manager available, the certificate and private key for `Ingress` are automatically requested by the `Ingress`. You only need to ensure that you have a valid `Issuer` available in cert-manager. You specify the name of the `Issuer` resource in a later step.
- If you are managing keys manually, create a certificate and private key and save it to a `Secret`. Take note of the `Secret` name, you will need to specify it in a later step.

This `Secret` referred to as `[***INGRESS_TLS_CERT_SECRET***]` in the following steps.

6. Set up resources for OAuth authentication and authorization.

- a) Generate a Java truststore (PKCS12) containing the TLS certificate of the root Certificate Authority (CA) of the OAuth certificate chain.

```
keytool -import -trustcacerts -file [***OAUTH ROOT CA***] \
  -keystore [***TRUSTSTORE NAME***] \
  -storepass [***TRUSTSTORE PASSWORD***] \
  -storetype PKCS12
```

- b) Create a Secret containing the truststore and its password.

```
kubectl create secret generic [***OAUTH TRUSTSTORE SECRET NAME***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***OAUTH TRUSTSTORE SECRET KEY***]=[***TRUSTSTORE
NAME***] \
  --from-file=[***OAUTH TRUSTSTORE PASSWORD SECRET KEY***]=[***PATH TO
TRUSTSTORE PW FILE***]
```

Take note of [***OAUTH TRUSTSTORE SECRET NAME***], [***OAUTH TRUSTSTORE SECRET KEY***], and [***OAUTH TRUSTSTORE PASSWORD SECRET KEY***].

7. Prepare Secrets for required PostgreSQL connection values.

Typically, you will need a Secret containing the PostgreSQL database password. Depending on your security requirements, an additional Secret might be required to hold TLS-related files, such as a Certificate Authority (CA) certificate used to establish a secure database connection.

- a) Create a Secret containing the PostgreSQL server password.

```
kubectl create secret generic [***POSTGRES SQL PASSWORD SECRET NAME***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***POSTGRES SQL PASSWORD SECRET KEY***]=[***PATH TO
DATABASE PASSWORD FILE***]
```

- b) Create a Secret containing TLS configuration files.

All files included in this Secret will be mounted to a common directory in the Pod, allowing the files to be referenced in your JDBC URL. In this example, a single CA certificate is added.

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

8. Prepare a custom values file (values.yaml).

The following example configures a fully secure deployment with a PostgreSQL database for persistent schema storage.

```
tls:
  enabled: true
  keystore:
    secretKeyRef:
      name: [***KEYSTORE SECRET NAME***]
      key: [***KEYSTORE SECRET KEY***]
  password:
    secretKeyRef:
      name: [***KEYSTORE SECRET NAME***]
      key: [***KEYSTORE PASSWORD SECRET KEY***]
  type: PKCS12

ingress:
  enabled: true
  className: "nginx"
  rules:
    path: "/"
```

```

host: "my-domain.example.com"
tls:
  enabled: true
  secretRef: [***INGRESS TLS CERT SECRET***]
extraAnnotations:
  nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

authentication:
  oauth:
    enabled: true
    jwt:
      principalClaimName: [***OAUTH PRINCIPAL CLAIM***]
      expectedAudience: [***OAUTH EXPECTED AUDIENCE***]
    jwks:
      url: [***OAUTH JWKS URL***]
      tls:
        truststore:
          secretKeyRef:
            name: [***OAUTH TRUSTSTORE SECRET NAME***]
            key: [***OAUTH TRUSTSTORE SECRET KEY***]
          password:
            secretKeyRef:
              name: [***OAUTH TRUSTSTORE SECRET NAME***]
              key: [***OAUTH TRUSTSTORE PASSWORD SECRET KEY***]
            type: PKCS12
  authorization:
    simple:
      enabled: true
      adminUsers: [***ADMIN USERS***]
      readOnlyUsers: [***READ-ONLY USERS***]

database:
  type: postgresql
  jdbcUrl: [***POSTGRESQL JDBC URL***]
  username: [***POSTGRESQL USERNAME***]
  password:
    secretKeyRef:
      name: [***POSTGRESQL PASSWORD SECRET NAME***]
      key: [***POSTGRESQL PASSWORD SECRET KEY***]
  tls:
    secretRef: [***POSTGRESQL TRUSTSTORE SECRET NAME***]

```

For tls

- `tls.enabled` – Enables or disables TLS.
- `tls.keystore.secretKeyRef.name` – The name of the `Secret` containing the TLS keystore.
- `tls.keystore.secretKeyRef.key` – The key in the `Secret` specified by `tls.keystore.secretKeyRef.name` that contains the TLS keystore.
- `tls.keystore.password.secretKeyRef.name` – The name of the `Secret` containing the TLS keystore password.
- `tls.keystore.password.secretKeyRef.key` – The key in the `Secret` specified by `tls.keystore.password.secretKeyRef.name` that contains the TLS keystore password.

For ingress

- `ingress.enabled` – Enables or disables external access through Ingress.
- `ingress.tls.enabled` – Enables or disables TLS for Ingress.
- `ingress.tls.secretRef` – The name of the `Secret` containing Ingress TLS certificates.
- `ingress.extraAnnotations.*` – Extra annotations to apply to the Ingress.



Note: If you are using cert-manager, add the cert-manager.io/issuer: *[**ISSUER NAME**]* annotation to the ingress.extraAnnotations property. If this annotation is set, a certificate is requested automatically and saved to the `Secret` specified in ingress.tls.secretRef.

For authentication

- authentication.oauth.enabled – Enables OAuth authentication for the Schema Registry server.
- authentication.oauth.jwt.principalClaimName – The name of the claim in the JWT token that contains the principal (username) used for authorization.
- authentication.oauth.jwt.expectedAudience – The expected audience value. If the JWT token contains an aud claim, it must match this value, otherwise the token is considered invalid.
- authentication.oauth.jwks.url – The URL to the JWKS endpoint.
- authentication.oauth.jwks.tls.truststore.secretKeyRef.name – The name of the `Secret` that contains the truststore for accessing the JWKS endpoint. Configure this property if the backend of your JWKS has self-signed certificates.
- authentication.oauth.jwks.tls.truststore.secretKeyRef.key – The key in the `Secret` specified by authentication.oauth.jwks.tls.truststore.secretKeyRef.name that contains the truststore for accessing the JWKS endpoint.
- authentication.oauth.jwks.tls.truststore.password.secretKeyRef.name – The name of the `Secret` that contains the truststore password for accessing the JWKS endpoint.
- authentication.oauth.jwks.tls.truststore.password.secretKeyRef.key – The key in the `Secret` specified by authentication.oauth.jwks.tls.truststore.password.secretKeyRef.name that contains the truststore password for accessing the JWKS endpoint.

For authorization

- authorization.simple.enabled – Enables or disables authorization.
- authorization.simple.adminUsers – A list of admin usernames. Admin users can perform any operation in Schema Registry.
- authorization.simple.readOnlyUsers – A list of read-only usernames. Read-only users can only perform read operations in Schema Registry.

For database

- database.jdbcUrl – The JDBC URL that points to your PostgreSQL database.
- database.username – The PostgreSQL username for Schema Registry database connections.
- database.password.secretKeyRef.name – The name of the `Secret` containing the PostgreSQL database password.
- database.password.secretKeyRef.key – The key in the `Secret` specified by database.password.secretKeyRef.name that contains the PostgreSQL database password.
- database.tls.secretRef – The name of a `Secret` containing TLS configuration for PostgreSQL connections (certificates, truststores, and so on). All keys from the `Secret` are mounted to `/etc/schema-registry/postgres/tls`. Reference mounted files in your JDBC URL (database.jdbcUrl) to configure SSL connections if SSL is required for PostgreSQL.

9. Install Schema Registry with helm install.

```
helm install schema-registry \
  --namespace [***NAMESPACE***] \
  --values [***VALUES FILE***] \
  --set image.registry=[***YOUR REGISTRY***] \
  --set 'image.imagePullSecrets=[***REGISTRY CREDENTIALS SECRET***]' \
  oci://[***YOUR REGISTRY***]/cloudera-helm/csm-operator/schema-registry \
  --version 1.7.0-b248
```

- The string schema-registry is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.

- `[[**VALUES FILE**]]` is the values file you prepared in Step 8 on page 45.
- `imagePullSecrets` specifies what `Secret` is used to pull images from the specified registry. Ensure that you replace `[[**REGISTRY CREDENTIALS SECRET**]]` with the name of the `Secret` you created in Step 3 on page 44.
- You can use `--set` to override properties that are defined in your values file, or add additional properties that are not present in your values file.

10. Verify your installation.

This is done by listing the `Deployments` and `Pods` in your namespace. If installation is successful, a `Schema Registry Deployment` and two `Pods` will be present in the cluster.

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

NAME		READY	UP-TO-DATE	AVAILABLE	AGE
#...					
schema-registry	2/2	2	2	13m	

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

NAME	READY	STATUS	RESTARTS	AGE
#...				
schema-registry-858f647cfc-82mkj	1/1	Running		0
13m				
schema-registry-858f647cfc-jl4nt	1/1	Running		0
13m				

What to do next

Configure clients to interact with Schema Registry or review and use the REST API.

Installing Schema Registry for evaluation

Complete these steps to install a basic deployment of Schema Registry that has no security configured and uses an in-memory database. Use these instructions if you want to install quickly in a development environment for proof of concept or evaluation purposes.

Before you begin

- Your Kubernetes environment meets requirements listed in [System requirements](#).
- Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [[**NAMESPACE**]]
```

This is the namespace where you install Schema Registry. Use the namespace you create in all installation steps that follow.

2. Log in to the Cloudera Docker registry with helm.

```
helm registry login container.repository.cloudera.com
```

Enter your Cloudera credentials when prompted.

3. Create a Kubernetes Secret containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
  -s password; echo >&2; echo $password)"
```

- Take note of the name you specify as [***REGISTRY CREDENTIALS SECRET***]. You will need to specify the name in a later step.
 - Replace [***USERNAME***] with your Cloudera username.
 - Enter your Cloudera password when prompted.
4. Prepare a custom values file (values.yaml).

The following example configures an unsecure deployment with an in-memory database.

```
tls:
  enabled: false

authentication:
  oauth:
    enabled: false

authorization:
  simple:
    enabled: false

database:
  type: in-memory

service:
  type: NodePort
```

- All security-related properties are set false to disable security. These properties must be explicitly set to false as the default value for all of them is true.
 - database.type – The type of database to use. The in-memory option starts Schema Registry with an ephemeral in-memory database that requires no additional configuration. However, in-memory mode is only suitable for testing and evaluation as all schemas will be lost when Pods restart.
 - service.type – The type of Kubernetes Service used for exposing the Schema Registry application. In this example NodePort is used instead of the default ClusterIP, so that Schema Registry is made accessible from outside the Kubernetes cluster.
5. Install Schema Registry with helm install.

```
helm install schema-registry \
  --namespace [***NAMESPACE***] \
  --values [***VALUES FILE***] \
  --set 'image.imagePullSecrets=[***REGISTRY CREDENTIALS SECRET***]' \
  oci://container.repository.cloudera.com/cloudera-helm/csm-operator/sch
  ema-registry \
  --version 1.7.0-b248
```

- The string `schema-registry` is the Helm release name of the chart installation. This is an arbitrary, user defined name. Cloudera recommends that you use a unique and easily identifiable name.
- `[***VALUES FILE***]` is the values file you prepared in Step 4 on page 49.
- `imagePullSecrets` specifies what `Secret` is used to pull images from the Cloudera registry. Setting this property is mandatory, otherwise, Helm cannot pull the necessary images from the Cloudera Docker registry. Ensure that you replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the `Secret` you created in Step 3 on page 49.
- You can use `--set` to override properties that are defined in your values file, or add additional properties that are not present in your values file.

6. Verify your installation.

This is done by listing the `Deployments` and `Pods` in your namespace. If installation is successful, a `Schema Registry Deployment` and two `Pods` will be present in the cluster.

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

NAME		READY	UP-TO-DATE	AVAILABLE	AGE
#...					
schema-registry	2/2	2	2	13m	

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

NAME	READY	STATUS	RESTARTS	AGE
#...				
schema-registry-858f647cfc-82mkj	1/1	Running		0
13m				
schema-registry-858f647cfc-jl4nt	1/1	Running		0
13m				

7. Access the Schema Registry UI.

Installing with `service.type: NodePort` deploys a `NodePort` type `Service` for Schema Registry making it accessible from any of the Kubernetes cluster nodes on the external port of the `Service`. List `Services` to get the external port.

```
kubectl get service schema-registry-service --namespace [***NAMESPACE***]
```

NAME (S)	AGE	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT
schema-registry-service		NodePort	10.43.121.112	<none>	9090
:31578/TCP	13m				

In this example, the external port is 31578.

What to do next

Configure clients to interact with Schema Registry or review and use the REST API.

Related Information

[Schema Registry REST API reference](#)

Installing Strimzi on Taikun CloudWorks [Technical Preview]

Learn how to install Strimzi in Cloudera Streams Messaging Operator for Kubernetes on Taikun CloudWorks. Installation involves importing the Cloudera Kafka Operator repository, adding Strimzi to a new or existing catalog, and installing Strimzi using the Taikun CloudWorks UI.

Before you begin



Note: This feature is in Technical Preview and is not ready for production deployments. Cloudera recommends trying this feature in test or development environments and encourages you to provide feedback on your experiences.

- You have access to a project and Kubernetes cluster in Taikun CloudWorks.
- Your Kubernetes environment meets requirements listed in [System requirements](#).
- The worker nodes in your cluster meet the minimum sizing requirements listed in [Sizing and performance considerations](#).
- Access to your cluster with kubectl is configured. For more information, see [Accessing Cluster with Kubeconfig](#).
- Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

- If you are planning to watch and manage more than 20 Kafka clusters with a single installation, you must increase the memory and heap allocated to the Strimzi Cluster Operator. For more information, see [Increasing Cluster Operator memory](#).

Importing the Cloudera Kafka Operator repository and adding Strimzi to a catalog

Complete these steps to import the Cloudera Kafka Operator repository and to add Strimzi to a new or existing catalog in Taikun CloudWorks.

Procedure

1. Import the Cloudera Kafka Operator repository.
 - a) In Taikun CloudWorks, go to Repositories and select the Private tab.
 - b) Click  Import Repository.
 - c) Enter the following in Import Repository:
 - Enter a unique name in Name.
 - Enter the following OCI repository URL in URL:
 - d) Click Import.

```
oci://container.repository.cloudera.com/cloudera-helm/csm-operator/s  
trimzi-kafka-operator
```

2. Add Strimzi to a catalog.



Tip: These instructions create a new catalog. You can also add your application to an existing catalog.

- a) Go to Catalogs and click  Add Catalog.
 - b) Enter a catalog name and description in Create Catalog.
 - c) Click Save.
 - d) Go to `[***YOUR CATALOG***]` and click  Add Applications.
 - e) Select `[***YOUR REPOSITORY***]` from the Repository drop-down list and click Apply.
 - f) Find the `strimzi-kafka-operator` application in the list of available applications and click .
 - g) Click  Add to the catalog.
- ## 3. Add catalog app parameters.
- a) Click Add Parameters.
 - b) Find and add the following parameters:
 - `watchAnyNamespaces`
 - `clouderaLicense.secretRef`
 - c) Set the following default values for the parameters you added:
 - `watchAnyNamespaces=true`
 - `clouderaLicense.secretRef=csm-op-license`
 - d) Click Save.

Installing Strimzi

Complete these steps to install Strimzi on Taikun CloudWorks.

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

Use this namespace in all of the following installation steps.

2. Create a Kubernetes Secret containing your Cloudera license.

```
kubectl create secret generic csm-op-license \
  --namespace [***NAMESPACE***] \
  --from-file=license=[***PATH TO LICENSE FILE***]
```

3. Create a Kubernetes Secret containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
-s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.

- Replace `[***USERNAME***]` with your Cloudera username.
 - Enter your Cloudera password when prompted.
4. Install Strimzi.
- a) In Taikun CloudWorks, go to Projects `[***YOUR PROJECT***]` Applications .
 - b) Click  Install.
 - c) Search for strimzi-kafka-operator.
 - d) Find the strimzi-kafka-operator application in the list of available applications. Select the one that is in `[***YOUR CATALOG ***]` and click .
 - e) Click Bind if you get a prompt to bind the catalog to your project.
 - f) Configure the following common settings in Application Instance:
 - Enter a name in Application Instance Name.
 - In Namespace, select the namespace you created in Step 1.
 - Enable the Extra Values tab by clicking the Extra Values toggle.
 - g) Click Continue.
 - h) Configure the following parameters in Installation Params:
 - Set the watchAnyNamespace toggle to enabled.
 - Set clouderaLicense.secretRef to the name of the Secret you created in Step 2.
-  **Tip:** Required parameters might already use correct default values.
- i) Click Continue.
 - j) Provide the following values in Extra Values:

```
image:
  imagePullSecrets:
    - name: [***REGISTRY CREDENTIALS SECRET***]
```

Replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the Secret you created in Step 3.

- k) Validate your extra values by clicking Check extra values.
 - l) Click  Run installation.
5. Go to Projects `[***YOUR PROJECT***]` LiveOps to verify your installation.
- If installation is successful, a Strimzi Deployment and Pod will be present in the cluster in the installation namespace. These resources use the name you specified in Application Instance Name.

What to do next

- Deploy a Kafka cluster, see [Deploying Kafka](#).



Note: When deploying a Kafka cluster on Taikun CloudWorks, use ingress type listeners, set the Ingress class to taikun, and ensure that bootstrap and broker hostnames resolve to your cluster.

Related Information

[Project Creation | Taikun CloudWorks](#)

[Creating Kubernetes cluster | Taikun CloudWorks](#)

[Accessing Cluster with Kubeconfig | Taikun CloudWorks](#)

[Installing Applications | Taikun CloudWorks](#)

[LiveOps | Taikun CloudWorks](#)

Installing Cloudera Surveyor for Apache Kafka on Taikun CloudWorks [Technical Preview]

Learn how to install Cloudera Surveyor in Cloudera Streams Messaging Operator for Kubernetes on Taikun CloudWorks. Installation involves importing the Cloudera Surveyor repository, adding Cloudera Surveyor to a new or existing catalog, and installing Cloudera Surveyor using the Taikun CloudWorks UI.

Before you begin



Note: This feature is in Technical Preview and is not ready for production deployments. Cloudera recommends trying this feature in test or development environments and encourages you to provide feedback on your experiences.

- You have access to a project and Kubernetes cluster in Taikun CloudWorks.
- Your Kubernetes environment meets requirements listed in [System requirements](#).
- The worker nodes in your cluster meet the minimum sizing requirements listed in [Sizing and performance considerations](#).
- Access to your cluster with kubectl is configured. For more information, see [Accessing Cluster with Kubeconfig](#).
- Your Kubernetes cluster requires internet connectivity to complete these steps. It must be able to reach the Cloudera Docker registry.
- You have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Archive and Cloudera Docker registry where installation artifacts are hosted.
- You have access to a valid Cloudera license.
- Review the [Helm chart reference](#) before installation.

The Helm chart accepts various configuration properties that you can set during installation. Using these properties you can customize your installation.

Importing the Cloudera Surveyor repository and adding Cloudera Surveyor to a catalog

Complete these steps to import the Cloudera Surveyor repository and to add Cloudera Surveyor to a new or existing catalog in Taikun CloudWorks.

Procedure

1. Import the Cloudera Surveyor repository.
 - a) In Taikun CloudWorks, go to Repositories and select the Private tab.
 - b) Click  Import Repository.
 - c) Enter the following in Import Repository:
 - Enter a unique name in Name.
 - Enter the following OCI repository URL in URL:
 - d) Click Import.
2. Add Cloudera Surveyor to a catalog.

```
oci://container.repository.cloudera.com/cloudera-helm/csm-operator/surveyor
```

- Enter your Cloudera credentials in Username and Password.



Tip: These instructions create a new catalog. You can also add your application to an existing catalog.

- a) Go to Catalogs and click  Add Catalog.
 - b) Enter a catalog name and description in Create Catalog.
 - c) Click Save.
 - d) Go to [***YOUR CATALOG***] and click  Add Applications.
 - e) Select [***YOUR REPOSITORY***] from the Repository drop-down list and click Apply.
 - f) Find the surveyor application in the list of available applications and click .
 - g) Click  Add to the catalog.
3. Add catalog app parameters.
 - a) Click Add Parameters.
 - b) Find and add the clouderaLicense.secretRef parameter.
 - c) Set the clouderaLicense.secretRef parameter to csm-op-license.
 - d) Click Save.

Installing Cloudera Surveyor

Complete these steps to install Cloudera Surveyor on Taikun CloudWorks.

About this task

These instructions walk you through installing an unsecure instance of Cloudera Surveyor on Taikun CloudWorks. Use these instructions for quick installation in development environments for proof of concept and evaluation purposes.

While the security configuration of Cloudera Surveyor itself is not covered, the example configuration demonstrates how to register a Kafka cluster that has TLS encryption enabled.

Procedure

1. Create a namespace in your Kubernetes cluster.

```
kubectl create namespace [***NAMESPACE***]
```

Use this namespace in all of the following installation steps.

2. Create a Kubernetes Secret containing your Cloudera license.

```
kubectl create secret generic csm-op-license \
  --namespace [***NAMESPACE***] \
  --from-file=license=[***PATH TO LICENSE FILE***]
```

3. Create a Kubernetes Secret containing your Cloudera credentials.

```
kubectl create secret docker-registry [***REGISTRY CREDENTIALS SECRET***] \
  --namespace [***NAMESPACE***] \
  --docker-server container.repository.cloudera.com \
  --docker-username [***USERNAME***] \
  --docker-password "$(echo -n 'Enter your Cloudera password: ' >&2; read
  -s password; echo >&2; echo $password)"
```

- Take note of the name you specify as `[***REGISTRY CREDENTIALS SECRET***]`. You will need to specify the name in a later step.
- Replace `[***USERNAME***]` with your Cloudera username.
- Enter your Cloudera password when prompted.

4. Create Secrets for sensitive Kafka client configuration values.

Cloudera Surveyor connects to Kafka clusters as any other Kafka client and requires a client configuration. If the Kafka cluster is secured, the client configuration will include sensitive property values. Cloudera recommends that you store sensitive values in **Secrets**, mount the **Secrets** to the Cloudera Surveyor Container, and reference the values in your configuration instead of hard-coding them.

Typically, you need to create two **Secrets**. One contains the Kafka cluster truststore and password, and the other contains a JAAS configuration. The Kafka truststore must be in JKS or PKCS12 format.

```
kubectl create secret generic [***KAFKA TRUSTSTORE SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***KAFKA TRUSTSTORE KEY***]=[***PATH TO TRUSTSTORE***] \
  --from-file=[***KAFKA TRUSTSTORE PASSWORD KEY***]=[***PATH TO TRUSTSTORE
  PASSWORD FILE***]
```

```
kubectl create secret generic [***KAFKA SASL.JAAS.CONFIG SECRET***] \
  --namespace [***NAMESPACE***] \
  --from-file=[***KAFKA SASL.JAAS.CONFIG KEY***]=[***PATH TO KAFKA
  SASL.JAAS.CONFIG FILE***]
```

- Take note of the **Secret** names as well as the key names you configure. You will need to specify them in a later step.
- All key names like `[***KAFKA TRUSTSTORE KEY***]` or `[***KAFKA SASL.JAAS.CONFIG KEY***]` are arbitrary.
- `[***PATH TO KAFKA SASL.JAAS.CONFIG FILE***]` is a path to a file containing JAAS configuration similar to the following example:

```
org.apache.kafka.common.security.plain.PlainLoginModule required username="MY-USER" password="MY-PASSWORD";
```

The contents of this file are set in a later step as the value of the `sasl.jaas.config` Kafka client property for Cloudera Surveyor internal Kafka clients. Ensure that the format of the configuration is valid for the `sasl.jaas.config` property. That is, its a single line of configuration.

5. Install Cloudera Surveyor.

- In Taikun CloudWorks, go to Projects `[***YOUR PROJECT***]` Applications .
- Click  Install.
- Search for surveyor.
- Find the surveyor application in the list of available applications. Select the one that is in `[*** YOUR CATALOG ***]` and click .
- Click Bind if you get a prompt to bind the catalog to your project.
- Configure the following common settings in Application Instance:
 - Enter a name in Application Instance Name.
 - In Namespace, select the namespace you created in Step 1.
 - Enable the Extra Values tab by clicking the Extra Values toggle.
- Click Continue.
- In Installation Params, set `clouderaLicense.secretRef` to the name of the **Secret** you created in Step 2.



Tip: Required parameters might already use correct default values.

- i) Click Continue.
- j) Provide the following values in Extra Values:

```
image:
  imagePullSecrets: [***REGISTRY CREDENTIALS SECRET***]
clusterConfigs:
  clusters:
    - clusterName: [***CLUSTER NAME***]
      tags:
        - [***TAG1***]
        - [***TAG2***]
      bootstrapServers: [***BOOTSTRAP SERVERS***]
      commonClientConfig:
        security.protocol: "SSL"
        ssl.truststore.type: "pkcs12"
        ssl.truststore.location: "/opt/secrets/[***KAFKA TRUSTSTORE
SECRET***]/[***KAFKA TRUSTSTORE FILE***]"
        ssl.truststore.password: "\\${dir:/opt/secrets/[***KAFKA
TRUSTSTORE SECRET***]:[***KAFKA TRUSTSTORE PASSWORD FILE***]}"
        sasl.jaas.config: "\\${dir:/opt/secrets/[***KAFKA
SASL.JAAS.CONFIG SECRET***]:[***KAFKA SASL.JAAS.CONFIG FILE***]}"
        adminOperationTimeout: PT1M
        authorization:
          enabled: false
      secretsToMount:
        - create: false
          secretRef: [***KAFKA TRUSTSTORE SECRET***]
          items:
            - key: [***KAFKA TRUSTSTORE KEY***]
              path: [***KAFKA TRUSTSTORE FILE***]
            - key: [***KAFKA TRUSTSTORE PASSWORD KEY***]
              path: [***KAFKA TRUSTSTORE PASSWORD FILE***]
        - create: false
          secretRef: [***KAFKA SASL.JAAS.CONFIG SECRET***]
          items:
            - key: [***KAFKA SASL.JAAS.CONFIG KEY***]
              path: [***KAFKA SASL.JAAS.CONFIG FILE***]
      surveyorConfig:
        surveyor:
          authentication:
            enabled: false
      tlsConfigs:
        enabled: false
      ingress:
        enabled: true
        className: taikun
        extraAnnotations:
          nginx.ingress.kubernetes.io/backend-protocol: HTTP
      rules:
        host: "[***SUBDOMAIN***].[***DOMAIN NAME***]"
        port: 8080
      tls:
        enabled: false
```

For image

image contains all configuration settings for the container image. This can include the image repository, tag version, pull policy, and any required authentication Secrets for accessing the registry.

In this example, `image.imagePullSecrets` is set. This property specifies the `Secret` that contains your Cloudera credentials. These credentials are used to access the Cloudera Docker registry to pull required images. Replace `[***REGISTRY CREDENTIALS SECRET***]` with the name of the `Secret` you created in Step 3.

For `clusterConfigs`

`clusterConfigs` specifies the Kafka clusters that Cloudera Surveyor connects to. Clusters specified here are the ones that will be available on the UI for monitoring and management. For more information and additional examples, see [Registering Kafka clusters](#).

- `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.clusters[*].commonClientConfig` – Kafka client configuration properties applied to all clients for this cluster. Must contain upstream Kafka client properties as a map. The exact properties that you specify here depend on the security configuration of the Kafka cluster that you want to connect. This example specifies a Kafka cluster that uses PLAIN authentication with TLS.

Sensitive property values are referenced from `Secrets` instead of being hardcoded. `Secrets` containing sensitive properties are mounted using the `secretsToMount` property. References use Kafka `DirectoryConfigProvider` syntax.



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

- `clusterConfigs.clusters[*].authorization.enabled` – Enables or disables authorization for this cluster.

For `secretsToMount`

`secretsToMount` specifies the `Secrets` to mount in the Cloudera Surveyor Container. You use `secretsToMount` to mount the sensitive values required for Kafka client configuration specified in `clusterConfigs`. For more information and additional examples, see [Managing sensitive data in client configuration](#)

- `secretsToMount[*].create` – Specifies whether to create the `Secret`. Set to false in this example as the `Secrets` are assumed to already exist.
- `secretsToMount[*].secretRef` – The name of the `Secret` to mount.
- `secretsToMount[*].items[*].key` – The key in the `Secret` to mount.
- `secretsToMount[*].items[*].path` – The path where the item is mounted. The path is relative to `/opt/secrets/[***SECRET NAME***]/` in the Cloudera Surveyor Container.

For `ingress`

`ingress` enables the creation of an `Ingress`. The `Ingress` provides external access to the Cloudera Surveyor UI.

- `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.

Configure this property as follows:

- `[***SUBDOMAIN***]` – Arbitrary and unique subdomain or service name that identifies the application. For example: `surveyor`.

- **[***DOMAIN NAME***]** – A domain that resolves to the access IP of your Kubernetes cluster.

If you do not have DNS configured or do not know your domain, use a wildcard DNS service like sslip.io. If you use a wildcard DNS service, the value you enter must have the following format:

```
[***SUBDOMAIN***].[***ACCESS IP***].[***WILDCARD DNS SERVICE***]
```

For example:

```
surveyor.203.0.113.255.sslip.io
```

You can find the access IP of the cluster in Taikun CloudWorks by going to Projects **[***YOUR PROJECT***]**

k) Validate your extra values by clicking Check extra values.

l) Click  Run installation.

6. Go to Projects **[***YOUR PROJECT***]** LiveOps to verify your installation.

If installation is successful, a Cloudera Surveyor Deployment and two Pods will be present in the cluster in the installation namespace. These resources use the name you specified in Application Instance Name.

7. Access the Cloudera Surveyor UI.

You can access the UI through the host you configured in ingress.rules.host.

Results

Cloudera Surveyor is installed. You can now manage and monitor your Kafka clusters using the UI.

Related Information

[Registering Kafka clusters](#)

[Managing sensitive data in client configurations](#)

[Project Creation | Taikun CloudWorks](#)

[Creating Kubernetes cluster | Taikun CloudWorks](#)

[Accessing Cluster with Kubeconfig | Taikun CloudWorks](#)

[Installing Applications | Taikun CloudWorks](#)

[LiveOps | Taikun CloudWorks](#)