Cloudera Streaming Analytics Operator 1.4.0

CSA Operator Installation

Date published: 2024-06-15
Date modified: 2025-10-24

https://docs.cloudera.com/

Legal Notice

Contents

Installation overview

Get started with installing Cloudera Streaming Analytics - Kubernetes Operator. Learn about the concept of installing, the installation artifacts, and where these artifacts are hosted.

Cloudera Streaming Analytics - Kubernetes Operator is installed using a Helm chart, which installs the Apache Kubernetes Flink Operator (Flink Operator), Cloudera SQL Stream Builder, and other components.

Installation artifacts and artifact locations

Note:

Cloudera Streaming Analytics - Kubernetes Operator operator offers alternative images that include the Hadoop, Hive, Iceberg, and Kudu connectors, as well as all required dependencies, in the Flink image.

To use the images with the connectors included, replace flink:1.20-csaop1.4.0-b40 with the name of the alternative image in the installation commands.

Cloudera Streaming Analytics - Kubernetes Operator ships with various installation artifacts, hosted at two locations: the Cloudera Docker registry and the Cloudera Archive.

Both the Cloudera Docker registry and the Cloudera Archive require your Cloudera credentials (username and password) to 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 Cloudera Docker registry hosts the Helm chart as well as all Docker images used for the installation.

Table 1: Cloudera Streaming Analytics - Kubernetes Operator artifacts on the Cloudera Docker registry

Artifact	Location (Base image)	Location (Image with connectors)	Description
Flink Kubernetes Operator Docker image	container.repository.cloudera.com/cloudera/flink-kubernetes-operator:1.11-csaop1.4.0-b55		Docker image used for deploying the various operator components shipped with the Cloudera Streaming Analytics - Kubernetes Operator.
Flink Docker image	container.repository.cloudera.com/cloudera/flink:1.20.1-csaop1.4.0-b55		Docker image used for deploying Apache Flink and its related components.
SQL Runner Docker image	container.repository.cloudera.com/cloudera/flink-extended:1.20.1-csaop1.4.0-b55	container.repository.cloudera.com/cloudera/flink-extended-hadoop:1.20.1-csaop1.4.0-b55	Docker image used for deploying Flink application when SQL query is executed in SSB.
SQL Stream Engine Docker image	container.repository.cloudera.com/cloudera/ssb-sse:1.20.1-csaop1.4.0-b55	container.repository.cloudera.com/cloudera/ssb-sse-hadoop:1.20.1-csaop1.4.0-b55	Docker image used for deploying Cloudera SQL Stream Builder and its UI.

Note: The images listed in the column Image with connectors contain the following connectors: Hadoop, Hive, Iceberg, and Kudu.

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

Cloudera Archive – archive.cloudera.com/p/csa-operator/1.4.0/

The Cloudera Archive hosts various installation artifacts including the Helm chart, diagnostic tools, and the Maven artifacts.

All artifacts hosted in the Cloudera Archive are supplemental resources, accessing them is not required to complete the installation. The following table collects the Cloudera Streaming Analytics - Kubernetes Operator directories located in the Cloudera Archive, with an overview of what artifacts they contain and how you can use them:

Table 2: CSA Operator artifacts on the Cloudera Archive

Archive Directory	Description
archive.cloudera.com/p/csa-operator/1.4.0/**charts**/	The charts directory contains the Helm chart. This is the same chart that is available in the Cloudera Docker registry. Cloudera recommends that whenever possible you install with the chart hosted in the registry. The chart in the archive is provided in case you cannot access the registry or want to download the chart using a browser.
archive.cloudera.com/p/csa-operator/1.4.0/**maven-repository**/	The Maven artifacts can be used to develop your own applications or tools for use with Cloudera Streaming Analytics - Kubernetes Operator.
archive.cloudera.com/p/csa-operator/1.4.0/**tools**/	The tools directory contains command line tools that you use to collect diagnostic information and to troubleshoot cluster issues.

Installing Cloudera Streaming Analytics - Kubernetes Operator in an environment with internet access

Complete these steps to install Cloudera Streaming Analytics - Kubernetes Operator if your Kubernetes cluster has internet access. Installing the Cloudera Streaming Analytics - Kubernetes Operator enables you to deploy and manage Flink and Cloudera SQL Stream Builder in Kubernetes.

About this task

Cloudera Streaming Analytics - Kubernetes Operator is installed in your Kubernetes cluster with the provided Helm chart through the helm install command. When you install the chart, Helm installs the Custom Resource Descriptors (CRD) included in Cloudera Streaming Analytics - Kubernetes Operator, and deploys the Apache Kubernetes Flink Operator (Flink Operator), Cloudera SQL Stream Builder engine, and a PostgreSQL database for Cloudera SQL Stream Builder.

Installing Cloudera Streaming Analytics - Kubernetes Operator does not create or deploy a Flink cluster. The Flink cluster is created after the installation by deploying the Flink Deployment resource in the Kubernetes cluster with kubectl or oc, or when you execute a SQL job in Streaming SQL Console.

Note: Cloudera recommends that you install Cloudera Streaming Analytics - Kubernetes Operator once per Kubernetes cluster.

By default the Flink Operator (deployed with installation) watches and manages all the Flink clusters that are deployed in the single same namespace as the Flink Operator. However, you can also configure it to watch and manage multiple namespaces. This allows you to manage multiple Flink clusters deployed in different namespaces, using a single Cloudera Streaming Analytics - Kubernetes Operator installation.

Important: Cloudera SQL Stream Builder can only be deployed in one namespace. In case you require multiple instances, you must install Cloudera SQL Stream Builder in every namespace. For more information about deploying Flink and Cloudera SQL Stream Builder in multiple namespaces, see the Namespace management documentation.

Before you begin

- Ensure that your Kubernetes environment meets requirements listed in System requirements.

  Note:

  If you're installing Cloudera Streaming Analytics - Kubernetes Operator on a Longhorn storage provider (Rancher/RKE2) the following line has to be added to the Helm values.yaml file:

  ```
  ssb:
    database:
      pod:
        securityContext:
          fsGroup: 999
  ```

- Your Kubernetes cluster requires internet connectivity to complete these steps, as it must be able to reach the Cloudera Docker registry.
- Ensure that you have access to a valid Cloudera license.
- Ensure that you have access to your Cloudera credentials (username and password, that are provided together with the Cloudera license) required to access the Cloudera Docker registry (and, if needed, the Cloudera Archive), 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. You can customize your installation using these properties.
- If you want to use the Webhook of Flink Operator, ensure that you have cert-manager installed on your Kubernetes cluster, which you can install using the following command:

  ```
  kubectl create -f https://github.com/jetstack/cert-manager/releases/down
  load/v1.8.2/cert-manager.yaml
  kubectl wait -n cert-manager --for=condition=Available deployment --all
  ```

  - The webhook functionality is enabled by default. You can disable it using the following command, and skip the cert-manager installation:

    ```
    --set flink-kubernetes-operator.webhook.create=false
    ```

Procedure

1. Create a namespace in your Kubernetes cluster where you will install and use the Cloudera Streaming Analytics - Kubernetes Operator.

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

   This is the namespace where you install Flink and Cloudera SQL Stream Builder. Use this namespace you create in all installation steps that follow.
2. Create a Kubernetes secret to contain your Cloudera credentials.

   ```
   kubectl create secret docker-registry [***SECRET NAME***] \
       --docker-server container.repository.cloudera.com \
       --docker-username [***USERNAME***] \
       --docker-password [***PASSWORD***] \
   ```

```
    --namespace [***NAMESPACE***]
```

Ensure that the placeholders are replaced with your specific information:

a) Provide a desired name for *[***SECRET NAME***]*.
b) Replace *[***USERNAME***]* and *[***PASSWORD***]* with your Cloudera credentials.
c) Provide the same name for *[***NAMESPACE***]* that you created in the previous step.

3. Log in to Cloudera Docker registry with helm.

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

Enter your Cloudera credentials when prompted.

4. Install Cloudera Streaming Analytics - Kubernetes Operator with helm install.

```
helm install csa-operator --namespace [***NAMESPACE***] \
    --set 'flink-kubernetes-operator.imagePullSecrets[0].name=[***SECRET
 NAME***]' \
    --set 'ssb.sse.image.imagePullSecrets[0].name=[***SECRET NAME***]' \
    --set 'ssb.sqlRunner.image.imagePullSecrets[0].name=[***SECRET
 NAME***]' \
    --set-file flink-kubernetes-operator.clouderaLicense.fileCon
tent=[***PATH TO LICENSE FILE***] \
oci://container.repository.cloudera.com/cloudera-helm/csa-operator/csa-op
erator --version 1.4.0-b55
```

Important: When you install Cloudera Streaming Analytics - Kubernetes Operator, Cloudera SQL Stream Builder will also be installed by default. In case you want to skip installing it, add --set ssb.enabled=false to the helm install command.

Ensure that the placeholders are replaced with your specific information:

a) Provide the same name for *[***NAMESPACE***]* that you created in Step 1.
b) Provide the same name for *[***SECRET NAME***]* that you created in the previous step. imagePullSecrets specifies what secret is used to pull images from the Cloudera registry. Setting this property is mandatory, otherwise Helm will be unable to pull the necessary images from the Cloudera Docker registry.
c) Replace *[***PATH TO LICENSE FILE***]* with the full (absolute) path to your Cloudera license file. clouderaLicense.fileContent is used to register your license. When this property is set, a secret is generated that contains the license you specify. Setting this property is mandatory. The Cloudera Streaming Analytics - Kubernetes Operator will not function without a valid license.
d) You can use --set to set various other properties of the Helm chart. This enables you to customize your installation. For example, by default the Flink Operator has access to watch all namespaces. However, you can configure a list of specific namespaces to watch using watchNamespaces.. For example, in case you created multiple namespaces, you can configure the Flink Operator to only watch specific ones with --set flink-kubernetes-operator.watchNamespaces={*[***NAMESPACE1***], [***NAMESPACE2***]*}. For more information about deploying Flink and SSB in multiple namespaces, see the Namespace management documentation.

5. Check that the Flink Operator, and the Cloudera SQL Stream Builder engine with its database are running.

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

```
NAME                                READY   STATUS      RESTARTS    AGE
flink-kubernetes-operator           1/2     Running      0          7s
ssb-postgresql                      1/1     Running      0          7s
ssb-sse                             1/1     Running      0          7s
```

What to do next

After successfully installing the Cloudera Streaming Analytics - Kubernetes Operator, you can start using Flink and Cloudera SQL Stream Builder on Kubernetes. The Getting Started with Flink and Getting Started with Cloudera SQL Stream Builder guides can help you with the basic operations.

Install Cloudera Streaming Analytics - Kubernetes Operator in Taikun CloudWorks [Technical Preview]

Installing the Cloudera Streaming Analytics - Kubernetes Operator enables you to deploy and manage Flink and Cloudera SQL Stream Builder in Taikun CloudWorks.

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.

Confirm these prerequisites before you start installing Cloudera Streaming Analytics - Kubernetes Operator in a Taikun-managed Kubernetes environment.

- You have access to a project and Kubernetes cluster in Taikun CloudWorks.
- Your Kubernetes environment meets the following sizing requirements:
  - **master:**4 CPU, 16GB RAM
  - **bastion**4 CPU, 16GB RAM
  - **worker (2x):**4 CPU, 16GB RAM
- Access to the cluster with kubectl is configured. For details, see *Accessing Cluster with Kubeconfig*.
- A catalog is available that includes the csa-operator application. For guidance, review *Importing the Cloudera Streaming Analytics - Kubernetes Operator Helm chart into Taikun CloudWorks*.
- Your Kubernetes cluster has outbound internet access to reach the Cloudera Docker registry.
- You have access to your Cloudera 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 Streaming Analytics - Kubernetes Operator 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.
- Install the cert-manager in your project, if it's not already present, so that Cloudera Streaming Analytics - Kubernetes Operator webhooks can operate correctly.

Related Information

Installing applications in Taikun CloudWorks
Cloudera Streaming Analytics - Kubernetes Operator documentation
Helm chart reference
Accessing Cluster with Kubeconfig

Importing the Cloudera Streaming Analytics - Kubernetes Operator Helm chart into Taikun CloudWorks

Import the Cloudera Streaming Analytics - Kubernetes Operator Helm chart repository and add it to a catalog so it is available for project installations.

About this task

Use these steps to make the Cloudera Streaming Analytics - Kubernetes Operator Helm chart available in your Taikun CloudWorks environment.

Procedure

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

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

      .
   e) Enter your Cloudera credentials.
   f) Click Import.
2. Add Cloudera Streaming Analytics - Kubernetes Operator to a catalog.

   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 the catalog Name and Description.
   c) Click Save.
   d) Go to *[***YOUR CATALOG***]* and click + Add Applications.
   e) Select *[*** YOUR REPIOSITORY***]* from the Repository drop-down list and click Apply.
   f) Find the **Cloudera Streaming Analytics - Kubernetes Operator** 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:

      ```
      clouderaLicense.secretRef
      ```

   c) Set the following default values for the parameters you added:
      - clouderaLicense.secretRef = csa-op-license
   d) Click Save.

Installing Cloudera Streaming Analytics - Kubernetes Operator in Taikun CloudWorks

Deploy the Cloudera Streaming Analytics - Kubernetes Operator application to your Taikun project by using the Helm chart you imported.

About this task

These steps use the Taikun UI to install Cloudera Streaming Analytics - Kubernetes Operator after you published the Helm chart in a catalog and installed cert-manager.

Procedure

1. Create a namespace for the deployment.

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

   Use this namespace for all Cloudera Streaming Analytics - Kubernetes Operator resources.

2. Create a Kubernetes secret that stores your Cloudera license.

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

3. Create a Docker registry secret with 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. In Taikun CloudWorks, go to Projects*[***YOUR PROJECT***]*Applications.
5. Click + Install.
6. Find the csa-operator.

   Select the application from the catalog that includes the repository you added earlier.

7. Click Bind if you get a prompt to bind the catalog to your project.
8. Configure the following common settings in Application Instance.
   a) Enter a name for Application Instance Name.
   b) In Namespace, select the namespace you created in step 1
   c) Enable the Extra Values tab by clicking the Extra Values toggle.
   d) Click Continue.
9. Configure the following parameters in Installation Params:
   a) Set clouderaLicense.secretRef to the name of the Secret you created in Step 2.
   b) Click Continue.
10. Provide the following values in Extra Values:

```
flink-kubernetes-operator:
  watchNamespaces:
    - [***NAMESPACE***]
  imagePullSecrets:
    - name: [***REGISTRY CREDENTIALS SECRET***]
ssb:
  sqlRunner:
    image:
      imagePullSecrets:
        - name: [***REGISTRY CREDENTIALS SECRET***]
  sse:
    image:
      imagePullSecrets:
        - name: [***REGISTRY CREDENTIALS SECRET***]
```

```
    ingress:
      spec:
        ingressClassName: taikun
        rules:
          - host: ssb.[***YOUR CLUSTER IP***].sslip.io
            http:
              paths:
                - backend:
                    service:
                      name: ssb-sse
                      port:
                        name: sse
                  path: /
                  pathType: ImplementationSpecific
  database:
    image:
      tag: 16.9
    pod:
      securityContext:
        fsGroup: 999
```

Replace *[***REGISTRY CREDENTIALS SECRET***]* with the name of the secret you created in Step 3. Replace *[***YOUR CLUSTER IP***]* with the IP address of *[***YOUR PROJECT***]*.

11. Validate your extra values by clicking Check extra values.
12. Choose Run Installation to deploy the application.
13. Go to ProjectsLiveOps to verify your installation.
14. Go to https://ssb.*[***YOUR CLUSTER IP***]*.sslip.io (see *Taikun Ingress* for more details) in a browser and log in to Cloudera SQL Stream Builder.

Related Information

Taikun Ingress

Installing Cloudera Streaming Analytics - Kubernetes Operator in an air-gapped environment

Complete these steps to install Cloudera Streaming Analytics - Kubernetes Operator if your Kubernetes cluster does not have internet access, or if you want to install from a self-hosted registry. Installing the Cloudera Streaming Analytics - Kubernetes Operator enables you to deploy and manage Flink and Cloudera SQL Stream Builder in Kubernetes.

About this task

Cloudera Streaming Analytics - Kubernetes Operator is installed in your Kubernetes cluster with the provided Helm chart through the helm install command. When you install the chart, Helm installs the Custom Resource Descriptors (CRD) included in Cloudera Streaming Analytics - Kubernetes Operator, and deploys the Apache Kubernetes Flink Operator (Flink Operator), Cloudera SQL Stream Builder engine, and a PostgreSQL database for Cloudera SQL Stream Builder.

Installing Cloudera Streaming Analytics - Kubernetes Operator does not create or deploy a Flink cluster. The Flink cluster is created after the installation by deploying the Flink Deployment resource in the Kubernetes cluster with kubectl or oc, or when you execute a SQL job in Streaming SQL Console.

Note: Cloudera recommends that you install Cloudera Streaming Analytics - Kubernetes Operator once per Kubernetes cluster.

By default the Flink Operator (deployed with installation) watches and manages all the Flink clusters that are deployed in the single same namespace as the Flink Operator. However, you can also configure it to watch and manage multiple namespaces. This allows you to manage multiple Flink clusters deployed in different namespaces, using a single Cloudera Streaming Analytics - Kubernetes Operator installation.

Important: Cloudera SQL Stream Builder can only be deployed in one namespace. In case you require multiple instances, you must install Cloudera SQL Stream Builder in every namespace. For more information about deploying Flink and Cloudera SQL Stream Builder in multiple namespaces, see the Namespace management documentation.

Before you begin

- Ensure that your Kubernetes environment meets requirements listed in System requirements.

  Note:

  If you're installing Cloudera Streaming Analytics - Kubernetes Operator on a Longhorn storage provider (Rancher/RKE2) the following line has to be added to the Helm values.yaml file:

  ```
  ssb:
    database:
      pod:
        securityContext:
          fsGroup: 999
  ```

- A self-hosted Docker registry is required. Your registry must be accessible by your Kubernetes cluster.
- While the Kubernetes cluster does not need internet access in an air-gapped environment, the preparation steps to create the local (offline) repository, from which you can install Cloudera Streaming Analytics - Kubernetes Operator, require that you can download and move the artifacts hosted on the Cloudera Docker registry and Cloudera Archive.
- Access to docker or equivalent utility that you can use to pull and push images is required. The Cloudera-recommended way is using docker. Replace commands where necessary, if you use a different utility.
- Ensure that you have access to your Cloudera credentials (username and password). Credentials are required to access the Cloudera Docker registry (and, if needed, the Cloudera Archive) where installation artifacts are hosted.
- Ensure that 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 want to use the Webhook of Flink Operator, ensure that you have cert-manager installed on your Kubernetes cluster, which you can install using the following command:

  ```
  kubectl create -f https://github.com/jetstack/cert-manager/releases/down
  load/v1.8.2/cert-manager.yaml
  kubectl wait -n cert-manager --for=condition=Available deployment --all
  ```

  - The webhook functionality is enabled by default. You can disable it using the following command, and skip the cert-manager installation:

    ```
    --set flink-kubernetes-operator.webhook.create=false
    ```

Procedure

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

Table 3: Cloudera Streaming Analytics - Kubernetes Operator artifacts on the Cloudera Docker registry

Artifact	Location (Base image)	Location (Image with connectors)	Description
Flink Kubernetes Operator Docker image	container.repository.cloudera.com/cloudera/flink-kubernetes-operator:1.11-csaop1.4.0-b55		Docker image used for deploying the various operator components shipped with the Cloudera Streaming Analytics - Kubernetes Operator.
Flink Docker image	container.repository.cloudera.com/cloudera/flink:1.20.1-csaop1.4.0-b55		Docker image used for deploying Apache Flink and its related components.
SQL Runner Docker image	container.repository.cloudera.com/cloudera/flink-extended:1.20.1-csaop1.4.0-b55	container.repository.cloudera.com/cloudera/flink-extended-hadoop:1.20.1-csaop1.4.0-b55	Docker image used for deploying Flink application when SQL query is executed in SSB.
SQL Stream Engine Docker image	container.repository.cloudera.com/cloudera/ssb-sse:1.20.1-csaop1.4.0-b55	container.repository.cloudera.com/cloudera/ssb-sse-hadoop:1.20.1-csaop1.4.0-b55	Docker image used for deploying Cloudera SQL Stream Builder and its UI.

Note: The images listed in the column Image with connectors contain the following connectors: Hadoop, Hive, Iceberg, and Kudu.

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 it out depend on your environment and how you set up your registry. The following substeps demonstrate a basic workflow using docker and helm.

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) Pull the Docker images from the Cloudera Docker registry.

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

c) Pull the Cloudera Streaming Analytics - Kubernetes Operator Helm chart.

```
helm pull \
  oci://container.repository.cloudera.com/cloudera-helm/csa-operator/csa
-operator \
  --version 1.4.0-b55
```

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

```
docker tag \
[***ORIGINAL IMAGE TAG***] \
```

```
[***REGISTRY HOSTNAME***]:[***PORT***]/cloudera/[***IMAGE NAME***]:
[***VERSION***]
```

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

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

```
helm push \
csa-operator-1.4.0-b55.tgz \
oci://[***REGISTRY HOSTNAME***]:[***PORT***]/cloudera-helm/csa-operator/
```

2. Create a namespace in your Kubernetes cluster where you will install and use the Cloudera Streaming Analytics - Kubernetes Operator.

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

This is the namespace where you install Flink and Cloudera SQL Stream Builder. Use this namespace you create in all installation steps that follow.

3. Create a Kubernetes secret to credentials for your self-hosted registry.

```
kubectl create secret docker-registry [***SECRET NAME***] \
    --docker-server [***REGISTRY HOSTNAME***]:[***PORT***] \
    --docker-username [***USERNAME***] \
    --docker-password [***PASSWORD***] \
    --namespace [***NAMESPACE***]
```

Ensure that the placeholders are replaced with your specific information:

a) Provide a desired name for *[***SECRET NAME***]*.
b) Replace *[***REGISTRY HOSTNAME***]*:*[***PORT***]* with your self-hosted registry hostname and port.
c) Replace *[***USERNAME***]* and *[***PASSWORD***]* with your Cloudera credentials.
d) Provide the same name for *[***NAMESPACE***]* that you created in the previous step.

4. Log in to your self-hosted registry with helm.

```
helm registry login [***REGISTRY HOSTNAME***]:[***PORT***]
```

Enter your credentials when prompted.

5. Install Cloudera Streaming Analytics - Kubernetes Operator with helm install.

```
helm install csa-operator \
    --namespace [***NAMESPACE***] \
    --set 'flink-kubernetes-operator.image.repository=[***REGISTRY
 HOSTNAME***]:[***PORT***]/cloudera/[***IMAGE NAME***]'\
    --set 'ssb.sqlRunner.image.repository=[***REGISTRY
 HOSTNAME***]:[***PORT***]/cloudera/[***IMAGE NAME***]'\
    --set 'ssb.sse.image.repository=[***REGISTRY
 HOSTNAME***]:[***PORT***]/cloudera/[***IMAGE NAME***]'\
    --set 'flink-kubernetes-operator.imagePullSecrets[0].name=[***SECRET
 NAME***]' \
    --set 'ssb.sse.image.imagePullSecrets[0].name=[***SECRET NAME***]' \
    --set 'ssb.sqlRunner.image.imagePullSecrets[0].name=[***SECRET
 NAME***]' \
    --set-file flink-kubernetes-operator.clouderaLicense.fileCon
tent=[***PATH TO LICENSE FILE***] \
```

```
oci://[***REGISTRY HOSTNAME***]:[***PORT***]/cloudera-helm/csa-operator/
csa-operator --version 1.4.0-b55
```

Important: When you install Cloudera Streaming Analytics - Kubernetes Operator, Cloudera SQL Stream Builder will also be installed by default. In case you want to skip installing it, add --set ssb.enabled=false to the helm install command.

Ensure that the placeholders are replaced with your specific information:

a) Provide the same name for *[***NAMESPACE***]* that you created in Step 1.
b) Replace *[***REGISTRY HOSTNAME***]*:*[***PORT***]* with your self-hosted registry hostname and port.
c) Provide the same name for *[***SECRET NAME***]* that you created in the previous step. 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.
d) Replace *[***PATH TO LICENSE FILE***]* with the full (absolute) path to your Cloudera license file. clouderaLicense.fileContent is used to register your license. When this property is set, a secret is generated that contains the license you specify. Setting this property is mandatory. The Cloudera Streaming Analytics - Kubernetes Operator will not function without a valid license.
e) You can use --set to set various other properties of the Helm chart. This enables you to customize your installation. (For more information on the available properties, see Helm chart reference.) For example, by default the Flink Operator has access to watch all namespaces. However, you can configure a list of specific namespaces to watch using watchNamespaces.. For example, in case you created multiple namespaces, you can configure the Flink Operator to only watch specific ones with --set flink-kubernetes-operator.watchNamespaces={*[***NAMESPACE1***]*, *[***NAMESPACE2***]*}. For more information about deploying Flink and Cloudera SQL Stream Builder in multiple namespaces, see the Namespace management documentation.

6. Check that the Flink Operator, and the Cloudera SQL Stream Builder engine with its database are running.

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

```
NAME                                READY   STATUS      RESTARTS    AGE
flink-kubernetes-operator           1/2     Running      0          7s
ssb-postgresql                      1/1     Running      0          7s
ssb-sse                             1/1     Running      0          7s
```

What to do next

After successfully installing the Cloudera Streaming Analytics - Kubernetes Operator, you can start using Flink and Cloudera SQL Stream Builder on Kubernetes. The Getting Started with Flink and Getting Started with Cloudera SQL Stream Builder guides can help you with the basic operations.

Configuring Cloudera Streaming Analytics - Kubernetes Operator for FIPS mode

Run Cloudera Streaming Analytics - Kubernetes Operator on a FIPS-enabled OpenShift cluster by using FIPS-approved cryptographic providers and custom container images.

About this task

Deploying Cloudera Streaming Analytics - Kubernetes Operator on a FIPS-enabled environment requires manual preparation to ensure every component starts, operates correctly, and uses FIPS-approved cryptographic libraries.

This procedure targets FIPS-enabled Red Hat OpenShift clusters that run on FIPS-enabled Red Hat nodes. The base container images ship with Red Hat builds of OpenJDK.

The steps assume the following FIPS providers are available and supplied by your organization:

- com.safelogic.cryptocomply.jcajce.provider.CryptoComplyFipsProvider (ccj-3.0.2.1.jar)
- org.bouncycastle.jsse.provider.BouncyCastleJsseProvider (bctls.jar)

If you use different providers, adjust the configuration steps accordingly.

The manual workflow includes the following actions:

- Supplying the crypto provider JAR files
- Creating a modified java.security file
- Building custom container images and pushing them to a registry that your cluster can access
- Creating truststores in .bcfks format
- Applying updated manifests and Helm values to enable components to consume the new assets

The examples assume that Cloudera Streaming Analytics - Kubernetes Operator is installed in the flink namespace with a command similar to the following:

```
helm install -n flink --set flink-kubernetes-operator.watchNamespaces={flink
} -f values.yaml csa-operator helm/csa-operator
```

Your values.yaml file must at minimum provide license and image repository details, for example:

```
flink-kubernetes-operator:
  clouderaLicense:

ssb:
  sqlRunner:
```

Verify that you can push custom images to *[***YOUR-DOCKER-REGISTRY***]* and that the cluster can pull from it. Commands include the --platform linux/amd64 option to support building on a different architecture, such as macOS on Apple Silicon.

Before you begin

- Access to a FIPS-enabled Red Hat OpenShift cluster that runs on FIPS-enabled Red Hat nodes
- Availability of the ccj-3.0.2.1.jar and bctls.jar provider files
- Credentials for a Docker registry that the cluster can access
- Ability to run docker, kubectl, helm, keytool, and Cloudera SQL clients from an administrative workstation
- A valid Cloudera license and access to the values.yaml file used by your Helm deployment
- Cert-manager installed if you plan to keep the Flink Operator webhook enabled

Procedure

1. Prepare the FIPS security provider artifacts and update the Java security configuration for the Flink Kubernetes Operator.

   You create a working directory, copy the provider JARs, and fetch the baseline java.security file from the running operator pod.

   a) Create the workspace and copy the provider files.

   ```
   mkdir fips
   cd fips

   # Copy the provider JARs into this directory.
   ls
         bctls.jar       ccj-3.0.2.1.jar
   ```

   b) Retrieve the java.security file from the operator pod.

   ```
   kubectl -n flink get pods
   ```

```
NAME                                              READY   STATUS    RESTARTS
  AGE
flink-kubernetes-operator-5b74769568-lb7ts   2/2     Running   0
  14m

kubectl -n flink exec -it pod/flink-kubernetes-operator-5b74769568-lb
7ts -- find /etc -name java.security
/etc/java/java-11-openjdk/java-11-openjdk-11.0.25.0.9-7.el9.x86_64/conf/
security/java.security
      kubectl -n flink cp flink-kubernetes-operator-5b74769568-lb7ts:/e
tc/java/java-11-openjdk/java-11-openjdk-11.0.25.0.9-7.el9.x86_64/conf/se
curity/java.security .
```

c) Edit the security providers in java.security.

Add your provider classes to the top of the security.provider and fips.provider lists and renumber the remaining entries. Set fips.keystore.type to BCFKS.

```
# List of providers and their preference orders.
security.provider.1=com.safelogic.cryptocomply.jcajce.provider.CryptoCo
mplyFipsProvider
security.provider.2=org.bouncycastle.jsse.provider.BouncyCastleJsseProv
ider fips:CCJ
security.provider.3=SUN
security.provider.4=SunRsaSign
...

# Security providers used when FIPS mode support is active.
fips.provider.1=com.safelogic.cryptocomply.jcajce.provider.CryptoCompl
yFipsProvider
fips.provider.2=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider
 fips:CCJ
fips.provider.3=SunPKCS11 ${java.home}/conf/security/nss.fips.cfg
fips.provider.4=SUN
...

# Default keystore type used when global crypto-policies are set to FI
PS.
      fips.keystore.type=BCFKS
```

2. Build and push a custom flink-kubernetes-operator image that bundles the FIPS provider configuration.

You create a Dockerfile that copies the provider JARs and the customized java.security file into the image and publishes the image to your registry.

a) Create the Dockerfile.

```
cat <<'EOF' > Dockerfile-flink-kubernetes-operator-fips
FROM container.repository.cloudera.com/cloudera/flink-kubernetes-operato
r:1.20.1-csaopX.X.X-bXX
COPY --chown=flink:flink bctls.jar $CLOUDERA_LIB/bctls.jar
COPY --chown=flink:flink ccj-3.0.2.1.jar $CLOUDERA_LIB/ccj-3.0.2.1.jar
COPY --chown=flink:flink java.security $CLOUDERA_LIB/java.security

ENV FIPS_JARS="/opt/flink/cloudera/bctls.jar:/opt/flink/cloudera/ccj-
3.0.2.1.jar"
ENV FIPS_ARGS="-Djava.security.properties==/opt/flink/cloudera/java.sec
urity"
      EOF
```

b) Build and push the image.

```
docker build --platform linux/amd64 -f Dockerfile-flink-kubernetes-opera
tor-fips -t [***YOUR-DOCKER-REGISTRY***]/flink/flink-kubernetes-operator
:fips-1 .
      docker push [***YOUR-DOCKER-REGISTRY***]/flink/flink-kubernetes-
operator:fips-1
```

3. Convert the webhook keystore to BCFKS format and update the Kubernetes secret.

a) Retrieve the keystore password and the existing PKCS12 file.

```
cd flink-kubernetes-operator/fips

kubectl get secret -n flink flink-operator-webhook-secret -o jsonpath
='{.data.password}' | base64 -d
password1234

    kubectl get secret -n flink webhook-server-cert -o jsonpath='{.dat
a.keystore\.p12}' | base64 -d > keystore.p12
```

b) Review the PKCS12 keystore contents.

```
keytool -list -keystore keystore.p12 -storetype PKCS12 -storepass passwo
rd1234

Keystore type: PKCS12
Keystore provider: SUN
Your keystore contains 1 entry
    ...
```

c) Convert the keystore to BCFKS format.

```
keytool -importkeystore \
    -srckeystore keystore.p12 \
    /* Lines 186-192 omitted */ \
    -providerpath ccj-3.0.2.1.jar

Importing keystore keystore.p12 to keystore.bcfks...
Entry for alias 1 successfully imported.
    Import command completed:  1 entries successfully imported, 0 entri
es failed or cancelled
```

d) Validate the BCFKS keystore and update the secret with the new file.

```
keytool -list \
    -keystore keystore.bcfks \
    /* Lines 201-204 omitted */ \
    -providerpath ccj-3.0.2.1.jar

Keystore type: BCFKS
Keystore provider: CCJ
Your keystore contains 1 entry
...

kubectl patch secret -n flink webhook-server-cert \
        --patch="{\"data\": {\"keystore.bcfks\": \"$(cat keystore.bcfks
| base64)\"}}"
```

4. Update the flink-kubernetes-operator deployment to reference the custom image and BCFKS keystore.

   You edit the deployment to configure both containers to use the new image, adjust the environment variables for the webhook keystore, and mount the BCFKS file from the secret.

   ```
   kubectl edit -n flink deployment flink-kubernetes-operator
   ```

   - Set each container image to *[***YOUR-DOCKER-REGISTRY***]*/flink/flink-kubernetes-operator:fips-1.
   - Update WEBHOOK_KEYSTORE_FILE to keystore.bcfks and ensure the corresponding secret item exposes the file with that name. Set WEBHOOK_KEYSTORE_TYPE to bcfks if it is defined.
   - Reference the new keystore.bcfks key in the projected secret volume.

   After you save and exit, Kubernetes restarts the pod. Confirm that the webhook container logs report the BCFKS keystore.

   ```
   2025-07-15 08:57:22,325 o.a.f.k.o.s.ReloadableSslContext [INFO ] Creating
    keystore with type: bcfks
   2025-07-15 08:57:22,329 o.a.f.k.o.s.ReloadableSslContext [INFO ] Loading
    keystore from file: /certs/keystore.bcfks
   2025-07-15 08:57:22,718 o.a.f.k.o.s.ReloadableSslContext [INFO ] Initiali
   zing key manager with keystore and password
   Jul 15, 2025 8:57:23 AM org.bouncycastle.jsse.provider.PropertyUtils get
   StringSystemProperty
   INFO: Found string system property [java.home]: /usr/lib/jvm/java-11-op
   enjdk-11.0.25.0.9-3.el9.x86_64
   Jul 15, 2025 8:57:23 AM org.bouncycastle.jsse.provider.PropertyUtils getS
   tringSystemProperty
   INFO: Found string system property [javax.net.ssl.trustStoreType]: pkcs12
   Jul 15, 2025 8:57:23 AM org.bouncycastle.jsse.provider.PropertyUtils getSt
   ringSystemProperty
   INFO: Found string system property [java.home]: /usr/lib/jvm/java-11-open
   jdk-11.0.25.0.9-3.el9.x86_64
   Jul 15, 2025 8:57:23 AM org.bouncycastle.jsse.provider.PropertyUtils ge
   tStringSystemProperty
   INFO: Found string system property [javax.net.ssl.trustStoreType]: pkcs12
   2025-07-15 08:57:23,125 o.a.f.k.o.a.FlinkOperatorWebhook [INFO ] Keystore
    path is resolved to real filename: keystore.bcfks
   2025-07-15 08:57:23,131 o.a.f.k.o.f.FileSystemWatchService [INFO ] Start
   ing watching path: /certs
   2025-07-15 08:57:23,132 o.a.f.k.o.f.FileSystemWatchService [INFO ] Path is
    resolved to real path: /certs
     2025-07-15 08:57:23,429 o.a.f.k.o.a.FlinkOperatorWebhook [INFO ] Webhook
    listening at 0:0:0:0:0:0:0:0:9443
   ```

5. Build and push a custom flink-extended image.

   a) Create the Dockerfile.

   ```
   cat <<'EOF' > Dockerfile-flink-extended-fips
   FROM container.repository.cloudera.com/cloudera/flink-extended:1.20.1-c
   saopX.X.X-bXX

   COPY --chown=9999:0 bctls.jar $FLINK_HOME/lib/bctls.jar
   COPY --chown=9999:0 ccj-3.0.2.1.jar $FLINK_HOME/lib/ccj-3.0.2.1.jar
   COPY --chown=9999:0 java.security $FLINK_HOME/lib/java.security

   ENV FIPS_JARS="/opt/flink/lib/bctls.jar:/opt/flink/lib/ccj-3.0.2.1.jar"
   ENV FIPS_ARGS="-Djava.security.properties==/opt/flink/lib/java.security"
       EOF
   ```

b) Build and push the image.

```
docker build --platform linux/amd64 -f Dockerfile-flink-extended-fips -
t [***YOUR-DOCKER-REGISTRY***]/flink/flink-extended:fips-1 .
    docker push [***YOUR-DOCKER-REGISTRY***]/flink/flink-extended:fips-1
```

6. Build and push a custom ssb-sse image.

a) Create the Dockerfile.

```
cat <<'EOF' > Dockerfile-ssb-sse-fips
FROM container.repository.cloudera.com/cloudera/ssb-sse:1.20.1-csaopX.
X.X-bXX
COPY --chown=9999:0 bctls.jar $FLINK_HOME/lib/bctls.jar
COPY --chown=9999:0 ccj-3.0.2.1.jar $FLINK_HOME/lib/ccj-3.0.2.1.jar
COPY --chown=9999:0 java.security $FLINK_HOME/lib/java.security

ENV FIPS_JARS="/opt/flink/lib/bctls.jar:/opt/flink/lib/ccj-3.0.2.1.jar"
ENV FIPS_ARGS="-Djava.security.properties==/opt/flink/lib/java.security"
    EOF
```

b) Build and push the image.

```
docker build --platform linux/amd64 -f Dockerfile-ssb-sse-fips -
t [***YOUR-DOCKER-REGISTRY***]/flink/ssb-sse:fips-1 .
    docker push [***YOUR-DOCKER-REGISTRY***]/flink/ssb-sse:fips-1
```

7. Update Helm values to reference the FIPS images and redeploy the release.

You edit values.yaml to point to the custom image tags and run helm upgrade.

```
flink-kubernetes-operator:
  image:
    repository: [***YOUR-DOCKER-REGISTRY***]/flink/flink-kubernetes-oper
ator
    tag: fips-1

ssb:
  sqlRunner:
    image:
      repository: [***YOUR-DOCKER-REGISTRY***]/flink/flink-extended
      tag: fips-1
  sse:
    image:
      repository: [***YOUR-DOCKER-REGISTRY***]/flink/ssb-sse
      tag: fips-1
```

```
helm upgrade -n flink --set flink-kubernetes-operator.watchNamespaces={f
link} -f values.yaml csa-operator helm/csa-operator
```

Validate the deployment by running a simple SQL job.

```
DROP TABLE IF EXISTS datagen_table_1;
DROP TABLE IF EXISTS print_sink_1;

CREATE TABLE  `datagen_table_1` (
  `col_str` STRING,
  /* Lines 348-349 omitted */
  `col_ts` TIMESTAMP(3)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);
```

```
CREATE TABLE  `print_sink_1` (
  `col_str` STRING,
  /* Lines 357-358 omitted */
  `col_ts` TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

insert into print_sink_1 select * from datagen_table_1;
```

8. Configure a Kafka connection that uses a BCFKS truststore.

   You create a truststore from the Kafka cluster certificate, expose it through a secret, update Helm values, and validate sampling in the UI.

   a) Create the BCFKS truststore and verify its contents.

   ```
   keytool -importcert \
       -alias kafka \
       /* Lines 380-386 omitted */ \
       -providerpath ccj-3.0.2.1.jar

   keytool -list \
       -keystore kafka.truststore.bcfks \
       /* Lines 391-394 omitted */ \
       -providerpath ccj-3.0.2.1.jar
   ```

   b) Create the Kubernetes secret.

   ```
   kubectl -n flink create secret generic kafka-tls --from-file=kafka.trust
   store.bcfks
   ```

   c) Reference the secret in values.yaml and run helm upgrade.

   ```
   ssb:
     podVolumes:
       # Configure volume mounts that expose the kafka-tls secret.
   ```

   ```
   helm upgrade -n flink --set flink-kubernetes-operator.watchNamespaces={f
   link} -f values.yaml csa-operator helm/csa-operator
   ```

   d) Enable sampling in the Cloudera SQL Stream Builder UI.

   In the Cloudera SQL Stream Builder UI, navigate to **Configuration** > **Sampling** and set the following values:

   - **Enabled:** yes
   - **Brokers:** *[***YOUR-KAFKA-BROKER***]*
   - **Protocol:** SSL
   - **Custom TrustStore:** yes
   - **Kafka TrustStore:** /opt/flink/tls/kafka.truststore.bcfks
   - **Kafka TrustStore Type:** bcfks
   - **Kafka TrustStore Password:** changeit

   Select **Validate** to confirm the configuration, then choose **Save**.

   e) Test the Kafka connection with a simple job.

   ```
   DROP TABLE IF EXISTS kafka_1;
   DROP TABLE IF EXISTS print_sink_kafka_1;

   CREATE TABLE `kafka_1` (
     `id` INT,
   ```

```
  `name` STRING
) WITH (
  'connector' = 'kafka',
  /* Lines 457-464 omitted */
  'format' = 'json'
);
CREATE TABLE  `print_sink_kafka_1` (
  `id` INT,
  `name` STRING
) WITH (
  'connector' = 'print'
);

insert into print_sink_kafka_1 select * from kafka_1;
```

9. Configure secure access to S3-compatible storage with FIPS truststores.

You create a BCFKS truststore for the S3 endpoint, expose it, update Helm values, and configure SQL jobs to use the truststore.

a) Create and validate the truststore.

```
keytool -importcert \
    -alias s3 \
    /* Lines 490-496 omitted */ \
    -providerpath ccj-3.0.2.1.jar

keytool -list \
    -keystore truststore.bcfks \
    /* Lines 501-504 omitted */ \
    -providerpath ccj-3.0.2.1.jar
```

b) Create the Kubernetes secret.

```
kubectl create secret generic truststore -n flink --from-file=truststore
.bcfks
```

c) Add storage configuration values and redeploy.

```
ssb:
  storageConfiguration:
    # Provide S3 access properties and mount the truststore secret.
```

```
helm upgrade -n flink --set flink-kubernetes-operator.watchNamespaces={f
link} -f values.yaml csa-operator helm/csa-operator
```

d) Run a job that sets the Java truststore properties globally.

```
DROP TABLE IF EXISTS `datagen_table_2`;
DROP TABLE IF EXISTS `print_sink_2`;

CREATE TABLE `datagen_table_2` (
  `col_str` STRING,
  /* Lines 549-550 omitted */
  `col_ts` TIMESTAMP(3)
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1'
);

CREATE TABLE `print_sink_2` (
  `col_str` STRING,
  /* Lines 558-559 omitted */
```

```
  `col_ts` TIMESTAMP(3)
) WITH (
  'connector' = 'print'
);

SET 'high-availability.type' = 'kubernetes';
SET 'high-availability.storageDir' = 's3://[***BUCKET***]/flink/recov
ery';
SET 'state.checkpoints.dir' = 's3://[***BUCKET***]/flink/checkpoints';
SET 'state.savepoints.dir' = 's3://[***BUCKET***]/flink/savepoints';
SET 'execution.checkpointing.interval' = '3s';
SET 'env.java.opts.all' = '-Djavax.net.ssl.trustStore=/opt/flink/tls/
truststore.bcfks -Djavax.net.ssl.trustStoreType=bcfks -Djavax.net.ssl.tr
ustStorePassword=changeit';
insert into print_sink_2 select * from datagen_table_2;
```

Note: Red Hat builds of OpenJDK set fips.keystore.type=PKCS11 by default. When the keystore type is PKCS11, the keystore path becomes NONE, which causes the S3 connector to fail because it treats NONE as a file name (JDK-8238264). Setting fips.keystore.type=BCFKS in java.security prevents this issue.

10. Configure LDAP authentication with a BCFKS truststore.

a) Create and verify the LDAP truststore.

```
keytool -importcert \
    -alias ldap \
    /* Lines 589-595 omitted */ \
    -providerpath ccj-3.0.2.1.jar
keytool -list \
    -keystore ldap_truststore.bcfks \
    /* Lines 600-603 omitted */ \
    -providerpath ccj-3.0.2.1.jar
```

b) Create the secret that stores LDAP connection properties.

```
kubectl create secret generic ssb-ldap -n flink \
  --from-literal=SSB_LDAP_URL=ldaps://[***YOUR-LDAP-SERVER***]:636 \
  /* Lines 611-621 omitted */ \
  --from-literal=SSB_LDAP_SSL_TRUSTSTORE_TYPE=bcfks
```

c) Update values.yaml and redeploy.

```
ssb:
  userManagement:
    # Reference the ssb-ldap secret and enable LDAP authentication.
```

```
helm upgrade -n flink --set flink-kubernetes-operator.watchNamespaces={f
link} -f values.yaml csa-operator helm/csa-operator
```

11. Use JDBC and CDC connectors with PostgreSQL over SSL.

a) Create a secret that contains the PostgreSQL CA certificate.

```
kubectl create secret generic postgresql-ssl -n flink --from-file=ca.crt
=postgres-ca.crt
```

b) Mount the secret in values.yaml.

```
ssb:
  podVolumes:
    # Reference the postgresql-ssl secret so Flink jobs can read ca.crt.
```

c) Configure PostgreSQL for logical decoding.

```
psql -U [***USERNAME***] -d [***DATABASE***] -c "ALTER SYSTEM SET wal_le
vel = logical;"
psql -U [***USERNAME***] -d [***DATABASE***] -c "SHOW wal_level;"
```

d) Run a JDBC example job.

```
DROP TABLE IF EXISTS jdbc_projects;
DROP TABLE IF EXISTS print_sink_jdbc;

CREATE TABLE jdbc_projects (
  id VARCHAR NOT NULL,
  /* Lines 679-682 omitted */
  active_environment_id INT
) WITH (
  'connector'   = 'jdbc',
  /* Lines 685-688 omitted */
  'password'    = '[***PASSWORD***]'
);

CREATE TABLE  `print_sink_jdbc` (
  id VARCHAR NOT NULL,
  /* Lines 693-696 omitted */
  active_environment_id INT
) WITH (
  'connector' = 'print'
);

INSERT INTO print_sink_jdbc SELECT * from jdbc_projects;
```

e) Run a CDC example job.

```
DROP TABLE IF EXISTS projects_cdc;
DROP TABLE IF EXISTS cdc_print;

CREATE TABLE projects_cdc (
  `id` VARCHAR(32),
  /* Lines 719-722 omitted */
  `active_environment_id` INT
) WITH (
  'connector' = 'postgres-cdc',
  /* Lines 725-734 omitted */
  'debezium.properties.database.sslrootcert' = '/opt/flink/postgresql-
ssl/ca.crt'
);
CREATE TABLE `ssb`.`ssb_default`.`cdc_print` (
  `id` VARCHAR(32),
  /* Lines 739-742 omitted */
  `active_environment_id` INT
) WITH (
  'connector' = 'print'
);
INSERT INTO cdc_print SELECT * FROM projects_cdc;
```

12. Use JDBC and CDC connectors with MySQL over SSL.

a) Create and validate the MySQL truststore.

```
keytool -importcert \
  -alias mysql \
  /* Lines 758-764 omitted */ \
  -providerpath ccj-3.0.2.1.jar
```

```
keytool -list \
    -keystore mysql_truststore.bcfks \
    /* Lines 769-772 omitted */ \
    -providerpath ccj-3.0.2.1.jar
```

b) Create the Kubernetes secret.

```
kubectl create secret generic mysql-truststore -n flink --from-file=mysq
l_truststore.bcfks=mysql_truststore.bcfks
```

c) Update values.yaml.

```
ssb:
  podVolumes:
    # Reference the mysql-truststore secret for Flink jobs.
```

d) Create sample data in MySQL.

```
USE demo;
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255)
);
INSERT INTO users (name) VALUES
  ('Alice'),
  /* Lines 808-809 omitted */
  ('Charlie');
SELECT * from users;
```

e) Run the JDBC example job.

```
DROP TABLE IF EXISTS jdbc_projects;
DROP TABLE IF EXISTS print_sink_jdbc;

CREATE TABLE jdbc_projects (
  id INT NOT NULL,
  name VARCHAR
) WITH (
  'connector'   = 'jdbc',
  /* Lines 824-827 omitted */
  'password'    = 'password'
);

CREATE TABLE  `print_sink_jdbc` (
  id INT NOT NULL,
  name VARCHAR
) WITH (
  'connector' = 'print'
);
INSERT INTO print_sink_jdbc SELECT * from jdbc_projects;
```

f) Run the CDC example job and review the current limitation.

```
DROP TABLE IF EXISTS projects_cdc;
DROP TABLE IF EXISTS cdc_print;

CREATE TABLE projects_cdc (
  `id` INT,
  `name` VARCHAR(255)
) WITH (
  'connector' = 'mysql-cdc',
  /* Lines 851-860 omitted */
  'debezium.database.ssl.truststore.password' = 'changeit'
);
```

```
CREATE TABLE `ssb`.`ssb_default`.`cdc_print` (
  `id` INT,
  `name` VARCHAR(255)
) WITH (
  'connector' = 'print'
);

INSERT INTO cdc_print SELECT * FROM projects_cdc;
```

Warning: This CDC task currently fails with java.lang.RuntimeException: SplitFetcher thread 1 received unexpected exception while polling the records.