Cloudera Streams Messaging Operator for Kubernetes 1.7.0

Schema Registry Configuration

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

https://docs.cloudera.com/

Legal Notice

Contents

Schema Registry configuration overview

Get started with configuring Schema Registry. Learn about basic configuration practices using Helm and available configuration properties.

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

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

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

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

Configurable properties

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

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

Related Information

Schema Registry Helm chart configuration reference

Configuring external access in Schema Registry

Learn how you can configure Schema Registry to make it accessible from outside of your Kubernetes cluster.

When installing Schema Registry with default values, a `ClusterIP` type Kubernetes `Service` is deployed. This provides access to Schema Registry from within the Kubernetes cluster.

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

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

Configuring external access with Ingress

Learn how to configure external access to Schema Registry with Kubernetes Ingress.

Before you begin

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

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

Procedure

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

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

   ```
   #...
   ingress:
     enabled: true
     className: "nginx"
     rules:
       host: MY-APP.EXAMPLE.CLOUDERA.COM
     tls:
       enabled: true
       secretRef: "[***INGRESS TLS CERT SECRET***]"
     extraAnnotations:
       cert-manager.io/issuer: "[***ISSUER NAME***]"
   ```

   - ingress.enabled – Enables or disables external access through `Ingress`.
   - ingress.rules.host – Specifies the DNS hostname that the `Ingress` controller should match for incoming HTTP/HTTPS requests.
   - ingress.tls.enabled – Enables or disables TLS for `Ingress`.
   - ingress.tls.secretRef – The name of the `Secret` containing Ingress TLS certificates.

     When using cert-manager and the cert-manager.io/issuer annotation is set in the ingress.extraAnnotations property, a certificate is requested automatically and saved to the `Secret` specified here.
   - ingress.extraAnnotations.* – Extra annotations to apply to the `Ingress`.

     The cert-manager.io/issuer annotation specified here contains the name of the cert-manager `Issuer`. When set, a certificate is automatically requested by the `Ingress`.
3. Apply configuration changes.

   ```
   helm upgrade SCHEMA-REGISTRY [***CHART***] \
     --namespace [***NAMESPACE***] \
     --values [***VALUES.YAML***] \
     --reuse-values
   ```
4. Access Schema Registry from the Hostname/IP of the `Ingress`.

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

   ```
   NAME                CLASS    HOSTS                ADDRESS     PORTS
   ```

```
#...
schema-registry-ingress   nginx   my-app.example.cloudera.com  10.14.91.1
  80, 443
```

In this example, Schema Registry is accessed at my-app.example.cloudera.com.com:443.

Configuring external access with LoadBalancer

Learn how to configure external access to Schema Registry with a LoadBalancer type Service.

Before you begin

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

Procedure

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

```
#...
service:
  type: LoadBalancer
  port: 9090
```

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

2. Apply configuration changes.

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

3. Access Schema Registry from the Hostname/IP of the load balancer.

```
kubectl get service SCHEMA-REGISTRY-service --namespace [***NAMESPACE***]
```

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

```
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP        PORT
(S)
schema-registry-service  LoadBalancer   10.103.58.116   104.198.205.71
 9090:30219/TCP
```

In this example, Schema Registry is accessed at 104.198.205.71:30219.

Database configuration for Schema Registry

Schema Registry requires a storage backend to persist schema metadata, versions, and compatibility settings. You can choose between PostgreSQL for production deployments or an in-memory database for testing and development environments.

Configuring a PostgreSQL database for Schema Registry

Configure persistent database storage for production Schema Registry deployments using PostgreSQL.

About this task

PostgreSQL is the recommended database backend for production deployments of Schema Registry. It provides persistent storage, high availability, and scalability for schema metadata. Configuration is done using database properties. You must specify the database type, JDBC connection URL, username, as well as a database password. You also specify a TLS `Secret` to mount certificate files for encrypted connections.

Before you begin

- 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 *[***POSTGRESQL JDBC URL***]* in the following steps.
- Get a username that Schema Registry can use to connect to the PostgreSQL server. Referred to as *[***POSTGRESQL USERNAME***]*.

Procedure

1. 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 [***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***]
   ```

2. Configure database properties in a values file (values.yaml).

   ```
   #....
   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 TLS SECRET NAME***]
   ```

   - 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.

3. Apply configuration changes.

```
helm upgrade schema-registry [***CHART***] \
  --namespace [***NAMESPACE***] \
  --values values.yaml \
  --reuse-values
```

Results

The PostgreSQL database is configured. Schema Registry now uses the specified database to persistently store schemas.

PostgreSQL TLS configuration

Reference a Kubernetes Secret containing certificate files to enable and configure TLS encryption for PostgreSQL connections.

The database.tls.secretRef property specifies a `Secret` containing certificate files for TLS connections to PostgreSQL. When configured, all keys from the `Secret` are mounted to /etc/schema-registry/postgres/tls inside the Schema Registry `Pods`. You then reference mounted files in your JDBC URL (database.jdbcUrl) to enable TLS.

The `Secret` you specify in the database.tls.secretRef property can contain various types of TLS-related files that are needed to establish an encrypted connection. The exact files the `Secret` must contain depends on the security requirements of your PostgreSQL server. Typically the `Secret` must contain a Certificate Authority (CA) certificate, which is used to verify the PostgreSQL server's identity.

Once mounted, files are referenced in your JDBC URL to enable TLS. For example, if your `Secret` contains a CA certificate file named ca.crt, your JDBC URL will be similar to the following example:

```
jdbc:postgresql://my-postgres-host:5432/schema_registry?sslmode=verify-full&
sslrootcert=/etc/schema-registry/postgres/tls/ca.crt
```

Notice how the sslrootcert parameter points to /etc/schema-registry/postgres/tls/ca.crt. This ensures the JDBC driver can find the certificate provided by your `Secret` at its designated mount point within the filesystem of the `Pod`.

Configuring an in-memory database for Schema Registry

Configure and ephemeral in-memory database for development and testing environments.

About this task

The in-memory database is designed for testing, development, and demonstration purposes. It stores all schema metadata in memory, which means data is lost when Schema Registry `Pods` restart. Configuration is done by setting the database.type property to in-memory.

Procedure

1. Configure an in-memory database in your values.yaml.

```
#...
```

```
database:
  type: in-memory
```

2. Apply configuration changes.

```
helm upgrade schema-registry [***CHART***] \
  --namespace [***NAMESPACE***] \
  --values values.yaml \
  --reuse-values
```

Results

Schema Registry is configured to use an in-memory database for schema storage.