Cloudera Runtime 7.1.6

Integrating with Schema Registry

Date published: 2019-11-08
Date modified: 2021-03-03

CLOUDERA

Legal Notice

Contents

Integrating with NiFi

Understand the NiFi Record Based Processors and Controller Services

The RecordReader and RecordWriter Controller Services and Processors allow you to convert events from one file format (json, xml, csv, Avro) to another (json, xml, csv, Avro). These controller services use Schema Registry to fetch the schema for the event to perform this conversion.

NiFi includes the following RecordReader and RecordWriter processors:

- ConsumeKafkaRecord_0_10 1.2.0
- ConvertRecord
- PublishKafkaRecord_0_10
- PutDatabaseRecord
- QueryRecord
- SplitRecord

NiFi also includes the following Record based Controller Services:

- HortonworksSchemaRegistry
- AvroRecordSetWriter
- CSVRecordSetWriter
- FreeFormTextRecordSetWriter
- JsonRecordSetWriter
- ScriptedRecordSetWriter

Configuring Schema Registry instance in NiFi

Configuring Schema Registry in NiFi is essential for seamless communication between the two components. To enable Schema Registry to communicate with NiFi dataflows, you need to configure the HortonworksSchemaRegistry Controller Service with the Schema Registry instance URL.

Before you begin

You have already installed Schema Registry.

Procedure

1. Navigate to Controller Settings in the Global Menu and select the Controller Services tab.
2. Click the + icon to access the Add Controller Service dialog.
3. Search for HortonworksSchemaRegistry in the filter box and click Add.
4. Open the Configure Controller Service dialog by clicking the Edit icon.
5. Provide the Schema Registry URL with which you want NiFi to communicate and click Apply.
6. Enable HortonworksSchemaRegistry by clicking the Enable icon, selecting the Scope, and then clicking Enable again.

Adding and Configuring Record Reader and Writer Controller Services

NiFi provides Record Reader and Writer Controller Services to support record-based processing. These Controller Services are new services that allows you convert events from one type (JSON, XML, CSV, Avro) to another. These

Controller Services use the Schema Registry to fetch the schema for the event to do this conversion. Before using these new Controller Services, you must configure them for use with Schema Registry.

You can configure Controller Services either globally, before you have created a Process Group, or at any time, on a per-Process Group basis.

Steps for Adding Controller Services globally

1. To access Controller Services configuration dialog for gobal configuration, click the Global Menu at the top right of your canvas, and select Controller Settings.
2. Click the + icon to display the NiFi Settings dialog.
3. Use the Filter box to search for the Controller Service you want to add, select that service, and click Add.

Steps for Adding Controller Services Per Process Group

1. Click on your Process Group, and then right-click anyway on your canvas.
2. Click Configure to display the Process Group Configuration dialog.
3. Click the Controller Services tab, and then click + to display the Add Controller Service dialog.
4. Use the Filter box to search for the Controller Service you want to add, select that service, and click Add.

Steps for Configuring Record Reader and Writer Controller Services for Integration with Schema Registry

1. From the Process Group Configuration view, click the Edit icon from the right-hand column. This displays the Configure Controller Service dialog.
2. Click the Properties tab.
3. The *SCHEMA ACCESS STRATEGY* specifies how to obtain the schema using for interpreting FlowFile data. To ensure integration with Schema Registry, configure *SCHEMA ACCESS STRATEGY* with one of the following two values:

   - HWX Schema Reference Attributes – The NiFi FlowFile is given a set of 3 attributes to describe the schema:

     - schema.identifier
     - schema.version
     - schema.protocol.version

   - HWX Content-Encoded Schema Reference – Each NiFi FlowFile contains a reference to a schema stored in Schema Registry. The reference is encoded as a single byte indicating the protocol version, 8 bytes indicating the schema identifier and 4 bytes indicating the schema version.

4. The *SCHEMA WRITE STRATEGY* specifies how the schema for a record should be added to FlowFile data. To ensure integration with Schema Registry, configure *SCHEMA WRITE STRATEGY* with either *HWX SCHEMA REFERENCE ATTRIBUTES* or*HWX CONTENT-ENCODED SCHEA REFERENCE.*

Using Record-Enabled Processors

Record-enabled processors allow you to convert data between formats by specifying Controller Services for record reading and record writing. This streamlines your dataflows and improves overall performance.

Procedure

1. From the NiFi UI, drag the Processor icon onto your canvas to display the Add Processor dialog.
2. Use the Filter box to find the Processor you want to add. Available record-enabled processors are:

   - ConsumeKafkaRecord_0_10
   - ConvertRecord
   - PublishKafkaRecord_0_10
   - PutDatabaseRecord
   - QueryRecord
   - SplitRecord

3. Select the Processor you want, and click Add.

4. Right-click the Processor on the canvas, and select Configure to display the Configure Processor dialog.

5. Click the Properties tab and select a Controller Service value for the Record Reader and Record Writer values.

6. Click OK and then Apply.

Integrating with Kafka

You can integrate Schema Registry with Kafka in one of two ways, depending on your use case. If you have NiFi, you can use NiFi processors to integrate Schema Registry with Kafka. If you do not have NiFi, you can integrate your Kafka producer and consumer manually.

Integrate Kafka and Schema Registry using NiFi Processors

If your cluster has Schema Registry, NiFi, and Kafka, you can use NiFi processors to integrate Schema Registry with Kafka. First integrate NiFi with Schema Registry, build the NiFi dataflow, and then add and configure the necessary Kafka processors.

Procedure

1. Integrate NiFi with Schema Registry.

2. Build your NiFi dataflow.

3. At the point in your dataflow where you want to either consume from a Kafka topic, or publish to a Kafka topic, add one of the following two processors:

   - *CONSUMEKAFKARECORD_0_10*
   - *PUBLISHKAFKARECORD_0_10*

4. Configure your Kafka processor with the following information:

   - Kafka Brokers – Provide a comma-separated list of Kafka Brokers you want to use in your dataflow.
   - Topic Name – The name of the Kafka topic to which you want to publish or from which you want to consume data.
   - Record Reader – Provide the Controller Service you want to use to read incoming FlowFile records.
   - Record Writer – Provide the Controller Service you want to use to serialize record data before sending it to Kafka.

Integrate Kafka and Schema Registry

If you are running CDP without NiFi, integrate your Kafka producer and consumer manually. To do this you must add a dependency on the Schema Registry Serdes, and update the Kafka producer and Kafka consumer configuration files.

Steps to Add a Schema Registry Serdes Dependency

1. Add the following text to schema-registry-serdes:

```
<dependency>
      <groupId>com.hortonworks.registries</groupId>
      <artifactId>schema-registry-serdes</artifactId>
</dependency>
```

Steps to Integrate the Kafka Producer

1. Add the following text to the Kafka Producer configuration:

```
config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
```

```
config.putAll(Collections.singletonMap(SchemaRegistryClient.Configuration
.SCHEMA_REGISTRY_URL.name(), props.get(SCHEMA_REGISTRY_URL)));
config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.c
lass.getName());
config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, KafkaAvroSeria
lizer.class.getName());
```

2. Edit the above text with values for the following properties:

- schema.registry.url
- key.serializer
- value.serializer

Steps to Integrate the Kafka Consumer

1. Add the following text to the Kafka Consumer configuration:

```
config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
config.putAll(Collections.singletonMap(SchemaRegistryClient.Configuration
.SCHEMA_REGISTRY_URL.name(), props.get(SCHEMA_REGISTRY_URL)));
config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializ
er.class.getName());
config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDes
erializer.class.getName());
```

2. Edit the above text with values for the following properties:

- schema.registry.url
- key.deserializer
- value.deserializer

Steps to Customize Client Cache Duration

By default, Schema Registry clients are only able to cache the schema for 5 minutes. To customize the cache
duration, configure the cache properties.

1. Customize the values for the following properties:

- Schema Metadata Cache:

  ```
  schema.registry.client.schema.metadata.cache.expiry.interval.secs
  ```

  Expiry interval (in seconds) of an entry in schema metadata cache. Default is 300 seconds.
- Schema Version Cache:

  ```
  schema.registry.client.schema.version.cache.expiry.interval.secs
  ```

  Expiry interval (in seconds) of an entry in schema version cache. Default is 300 seconds.
- Schema Text Cache:

  ```
  schema.registry.client.schema.text.cache.expiry.interval.secs
  ```

  Expiry interval (in seconds) of an entry in schema text cache. Default is 300 seconds.
- Class Loader Cache

  ```
  schema.registry.client.class.loader.cache.expiry.interval.secs
  ```

  Expiry interval (in seconds) of an entry in the serializer/deserializer class loader cache. Default is 300 seconds.

Improve Performance in Schema Registry

You can increase the performance of Schema Registry by adding Apache Knox, the CDP load balancer, to your Schema Registry configuration. This increase in performance will offset the minor decrease in performance caused by the lack of server-side caching.

By default, Schema Registry does not have server-side caching. Each time a client sends a query, the client reads the data directly from the database.

The lack of server-side caching creates a minor performance impact. However, it enables Schema Registry to be stateless and be distributed over multiple hosts. All hosts will use the same database. This enables a client to insert a schema on host A and query the same schema on host B.

You can store JAR files similarly in HDFS and retrieve a file even if one of the hosts is down.

To make Schema Registry highly available in this setup, include a load balancer on top of the services. Clients will communicate through the load balancer. The load balancer will then determine which host to send the requests to. If one host is down, the request is sent to another host. In CDP, the load balancer is Apache Knox.

The following diagram shows the flow of a client query and response: