FLIP-125: Confluent Schema Registry Catalog

- Status
- Motivation
- Introduction to Confluent Schema Registry
  - Terminology
  - Subject Naming Strategy
  - The Restful API
  - Security
  - Schema Compatibility
  - Schema & Format
    - Schema and Format Binding
    - Format Compatibility
- Design Proposal
  - SchemaRegistryCatalog
    - Parameters
    - Basic Auth Security for Producers and Consumers
  - RegistryAvroFormatFactory
  - Public Interfaces
  - The Expected Behaviors
  - The Table Schema and Watermark Definition
    - The Watermark Definition
    - The Key Fields as Part of the Schema
- Compatibility, Deprecation, and Migration Plan
- Implementation Plan
- Test Plan

Status

<table>
<thead>
<tr>
<th>Discussion thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vote thread</td>
</tr>
<tr>
<td>JIRA</td>
</tr>
</tbody>
</table>

Release 1.3

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Confluent Schema Registry provides a RESTful interface for developers to define standard schemas for their events, share them across the organization and safely evolve them in a way that is backward compatible and future proof. Schema Registry stores a versioned history of all schemas and allows the evolution of schemas according to the configured compatibility settings. It also provides a plugin to clients that handles schema storage and retrieval for messages that are sent in Avro format.

A Confluent Schema Registry Catalog make the Flink SQL table access extremely convenient, all need to config is a single schema registry URL, then all the Kafka topics registered in the schema registry service can be accessed in Flink SQL and table API. Here is a code snippet to illustrate how to access tables by registering such a catalog:

```java
String schemaRegistryURL = ...;
Map<String, String> kafkaProps = ...;
SchemaRegistryCatalog catalog = SchemaRegistryCatalog.builder()
    .schemaRegistryURL(schemaRegistryURL)
    .kafkaOptions(kafkaProps)
    .dbName("myDB")
    .build();
tEnv.registerCatalog("myCatalog", catalog);

// ---------- Consume stream from Kafka -------------------

// Assumes there is a topic named 'transactions'
String query = "SELECT\n" +
"  id, amount\n" +
"FROM myCatalog.myDB.transactions";
```
Introduction to Confluent Schema Registry

Terminology

What is a topic versus a schema versus a subject?

- Subject: Schema Registry defines a scope in which schemas can evolve, and that scope is the subject. The name of the subject depends on the configured subject name strategy, which by default is set to derive subject name from topic name.
- Topic: Kafka topic contains messages, and each message is a key-value pair. Either the message key or the message value, or both, can be serialized as Avro, JSON, or Protobuf.
- Schema name: Schema name, for Avro it is the record name, for JSON, it is the title name.

See [terminology-review](#) for details.

Subject Naming Strategy

There are 3 kinds of naming strategy for current 5.5.1 version:

- TopicNameStrategy: `<topic name>-key | <topic name>-value`
- RecordNameStrategy: `<fully-qualified record name>-key | <fully-qualified record name>-value`
- TopicRecordNameStrategy: `<topic name>-<fully-qualified record name>-key | <topic name>-<fully-qualified record name>-value`

The `RecordNameStrategy` allows different schemas (record type) within one Kafka topic. The schema compatibility check for same record name are among all the topics while `TopicRecordNameStrategy` checks the compatibility of same record name schema within one Kafka topic.

See [sr-schemas-subject-name-strategy](#) for details.

The Restful API

See [schemaregistry-api](#) for details.

security

- [https://docs.confluent.io/current/security/basic-auth.html#schema-registry](https://docs.confluent.io/current/security/basic-auth.html#schema-registry)
- [https://docs.confluent.io/current/schema-registry/security/index.html#sr-security-overview](https://docs.confluent.io/current/schema-registry/security/index.html#sr-security-overview)
- [https://docs.confluent.io/current/schema-registry/serdes-develop/index.html#basic-auth-security-for-producers-and-consumers](https://docs.confluent.io/current/schema-registry/serdes-develop/index.html#basic-auth-security-for-producers-and-consumers)

Schema Compatibility

From 5.5.0, Schema Registry allows different compatibility level per-subject or globally, the Schema Registry service would force compatibility check when the schema evolves.

- [https://docs.confluent.io/current/schema-registry/avro.html#compatibility-types](https://docs.confluent.io/current/schema-registry/avro.html#compatibility-types)
- [http://avro.apache.org/docs/1.7.7/spec.html#Schema+Resolution](http://avro.apache.org/docs/1.7.7/spec.html#Schema+Resolution)
- [https://docs.confluent.io/current/schema-registry/serdes-develop/serdes-json.html#json-schema-compatibility-rules](https://docs.confluent.io/current/schema-registry/serdes-develop/serdes-json.html#json-schema-compatibility-rules)
- [https://docs.confluent.io/current/schema-registry/develop/api.html#id1](https://docs.confluent.io/current/schema-registry/develop/api.html#id1)

Schema & Format

Schema and Format Binding

From 5.5.0, user can get format for a schema through `confluent SchemaRegistryClient#getLatestSchemaMetadata`, i.e. Avro, JSON or Protobuf.

Format Compatibility

The format also has its own compatibility rules, for example, for Avro: [http://avro.apache.org/docs/1.7.7/spec.html#Schema+Resolution](http://avro.apache.org/docs/1.7.7/spec.html#Schema+Resolution)

Design Proposal

Note: We only support avro format for this FLIP !!!

SchemaRegistryCatalog

The SchemaRegistryCatalog interacts with the Confluent Schema Registry Service directly through its java SDK client, e.g. the SchemaRegistryClient.

- when open/initialize, it fetches all the topic list of current Kafka cluster and makes a cache in the memory.
when a specific catalog table info is requested from the SQL context, it fetches the Avro schema string and format type of the topic's latest subject schema, then generates a catalog table instance and put it into the local cache.

We dropped the idea to have table cache because the cache would introduce in-consistency with the latest SR state.

A catalog table instance with the following attributes was generated each time #getTable is invoked:

- TableSchema: inferred from the Avro schema string
- Confluent schema registry URL
- The Avro schema string
- The schema registry subject name (actually with format {topic-name}-value, used for write)
- The Kafka topic name
- Format identifier "avro-sr", this helps to find formats for Confluent Schema Registry Se/De
- Common properties for all the topics (as a parameter of the Catalog)

The graph below illustrates how this catalog works with the Confluent Schema Registry Service:

1. Fetch table info from the catalog table cache
2. If it exists, return early
3. Else requests the schema registry service, put the table info into cache and returns

Parameters

The SchemaRegistryCatalog.Builder can be used to configure the following options:

<table>
<thead>
<tr>
<th>name</th>
<th>isOptional</th>
<th>default</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectorOptions</td>
<td>false</td>
<td>(null)</td>
<td>Connector options for all the tables read from this catalog, the options are shared for all the tables, if you want to tweak or override per-table scope, use the dynamic table options or CREATE TABLE LIKE syntax.</td>
</tr>
<tr>
<td>registryURL</td>
<td>false</td>
<td>(null)</td>
<td>Schema Registry URL to connect to the registry service.</td>
</tr>
<tr>
<td>dbName</td>
<td>true</td>
<td>kafka</td>
<td>database name</td>
</tr>
<tr>
<td>registryClient</td>
<td>true</td>
<td>CachedSchemaRegistryClient with default identityMapCapacity of 1000 as (code identityMapCapacity). Sets up the (link SchemaRegistryClient) to connect to the registry service. By default, the catalog holds a (link CachedSchemaRegistryClient) with 1000 as (code identityMapCapacity). This method is used for custom client configuration, i.e. the SSL configurations or to change the default (code identityMapCapacity).</td>
<td></td>
</tr>
</tbody>
</table>

Basic Auth Security for Producers and Consumers

See https://docs.confluent.io/current/schema-registry/serdes-develop/index.html#basic-auth-security-for-producers-and-consumers.
If you want to configure the client to enable SSL, use a custom SchemaRegistryClient when constructing the catalog.

**RegistryAvroFormatFactory**

RegistryAvroFormatFactory is a factory for Confluent Schema Registry Avro SerDe formats. It has the following attributes:

- `format`: avro-sr: factory ID, required
- `schema-registry.url`: schema registry URL, required
- `schema-string`: avro schema string, required
- `schema-registry.subject`: subject to write to, required only for sink table

**Public Interfaces**

```java
public class SchemaRegistryCatalog extends TableCatalog {
    public SchemaRegistryCatalog() {
    
    } // ---------- Consume stream from Kafka -------------------

    String query = "SELECT
    id, amount
FROM myCatalog.myDB.transactions";

    } // Only support TopicNameStrategy for subject naming strategy,
    // for which all the records in one topic has the same schema, see
    // How the Naming Strategies Work
    // downloaded from https://docs.confluent.io/current/schema-registry/serializer-formatter.html#how-the-naming-strategies-work
    // for details.

    // The behaviors:
    public class SchemaRegistryCatalog extends TableCatalog {
        public SchemaRegistryCatalog extends TableCatalog {
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
        // for which all the records in one topic has the same schema, see
        // How the Naming Strategies Work
        
        // The behaviors:
        
        } // Only support TopicNameStrategy for subject naming strategy,
ConfluentRegistryAvroRowFormatFactory

```java
/**
 * Table format factory for providing configured instances of Schema Registry Avro to RowData
 * `{@link SerializationSchema} and `{@link DeserializationSchema}.
 */
public class RegistryAvroFormatFactory implements
    DeserializationFormatFactory,
    SerializationFormatFactory {

}
```

The Expected Behaviors

- The catalog only supports reading messages with the latest enabled schema for any given Kafka topic at the time when the SQL query was compiled.
- It does not support time-column and watermark support.
- The catalog only supports Kafka message values prefixed with schema id, this is also the default behavior for the SchemaRegistry Kafka producer format.

The Table Schema and Watermark Definition

Table that reads from the ConfluentSchemaRegistryCatalog only has fields of the value part of the Kafka record which is with Avro format, for example, an Avro schema string:

```json
{"namespace": "io.confluent.examples.clients.basicavro",
 "type": "record",
 "name": "Payment",
 "fields": [
   {
     "name": "id",
     "type": "string"
   },
   {
     "name": "amount",
     "type": "double"
   }
 ]
}
```

would yield Table schema <id: STRING, amount: DOUBLE>, there is no fields that comes from the record key part and no water mark strategy definition.

The Watermark Definition

Base on the FLIP-110, user can use the LIKE clause to append a watermark definition to the table reading from the Catalog, for example:

```
CREATE [TEMPORARY] TABLE derived_table (  
   WATERMARK FOR tstmp AS tsmp - INTERVAL '5' SECOND
)
LIKE base_table; -- base_table comes from the ConfluentSchemaRegistryCatalog
```

The Key Fields as Part of the Schema

Base on the FLIP-110 and FLIP-107, user can use the LIKE clause to append key columns to the table reading from the Catalog, for example:

```
CREATE [TEMPORARY] TABLE derived_table (  
   id BIGINT,
   name STRING,
 ) WITH (  
   'key.fields' = 'id, name',
   'key.format.type' = 'csv'
)
LIKE base_table; -- base_table comes from the ConfluentSchemaRegistryCatalog
```

Compatibility, Deprecation, and Migration Plan
This is a new feature so there is no compatibility problem.

**Implementation Plan**

- Add a new Catalog named ConfluentSchemaRegistryCatalog
- Add a format factory ConfluentRegistryAvroRowFormatFactory
- Add two formats: ConfluentRegistryAvroRowDeserializationSchema and ConfluentRegistryAvroRowSerializationSchema

**Test Plan**

The Confluent Schema Registry is a service on Kafka cluster, we need a e2e test for both read and write of Kafka topics.