Adding Schema Registry Support to Metastore

The team building the Hortonworks Schema Registry has expressed interest in integrating it into the standalone metastore. This will allow users to have one data catalog for table based and stream based data. This document covers the work to the metastore to enable supporting the objects in the Schema Registry.

Data Model

The Metastore and the SR do not completely share a data model. The following describes what each currently has and changes necessary to bridge the gaps.

Mapping SR to Metastore

Schema Registry Data Model

The Schema Registry has a fairly small group of objects it stores. These include

- Schema - a named, versioned object, with SerDes mapped to it. Note that Schema does NOT include a set of typed, named fields. Schemas have policies on their compatibility between versions.
- SchemaVersion - a particular version of a particular Schema. This object does have the set of typed, named fields (e.g. a string, b int).
- SerDes - a pair of classes used to serialize and deserialize data described by a schema.

SR supports the following operations:

- Schema operations
    - Add
    - Update
    - Fetch
    - Fetch all versions of all schemas that match a query
    - Search schemas using a filter and return results in a particular sort order
- Schema version operations
    - Add a version
    - Delete a version
    - Fetch info about a version
    - Fetch info about the latest version
    - Get all versions
    - Check compatibility between a proposed schema and the latest version or all versions
    - Find all versions that contain a given column
    - Change state according to a defined state transition model
    - Find schema version by schema text
    - Get information about all versions of a schema

- SerDes operations
    - Add a SerDes
    - Map a Schema to a SerDes
    - Get all SerDes associated with a Schema

A Schema object has type (e.g. Avro, JSON), group (e.g. Hive, Kafka), name (must be unique), description, compatibility (one of: none, backward, forward, both), validation level (whether to validate against only the latest version or all versions), and whether the schema is allowed to evolve (e.g., have multiple versions).

A schema version has schema text (the JSON that describes the schema), a description, and an initial state (one of initiated, start_review, changes_required, reviewed, enabled, disabled, archived, or deleted).

SerDes have a unique id, a timestamp, a name, a description, a file id (the jar file), serializer class, and deserializer class.

Schema fields can be of type boolean, byte, short, integer, long, float, double, string, binary (backed by Java byte[]), nested (backed by Java Map), array, or blob (backed by Java InputStream). Schema fields have an id, a namespace, a name, and a type.

In the Schema Registry schema column lists are generally represented as Java Strings. Methods are provided to parse these Strings in order to get a collection of SchemaFieldInfo objects. This parsing depends on the schema type. The Metastore will need to make use of this to translate schema text into a list of schema fields.

Metastore Data Model

The metastore has an object called StorageDescriptor, which along with other information, contains the list of fields a table or partition has, and the SerDe used to access that table.

FieldSchema records the name, description, and type of each column. Columns can be of type: boolean, tiny int, small int, integer, big int, float, double, string, char, varchar, date, datetime, timestamp, decimal, binary, interval, list, map, struct, or union.

SerDeInfo records the class name of a SerDe, the jar the SerDe is in, and parameters describing the SerDe.

Connecting the Two Models

Eventually we would like to use the Metastore's StorageDescriptor as the base connection point between the two. The reason for this is eventually we would like the engines using these schemas to be able to interoperate. That is, Hive should be able to view a Kafka topic as a table and query it. SR users should be able to view a Hive table as a data source and read data from it. By tying all users together at the StorageDescriptor this becomes possible.

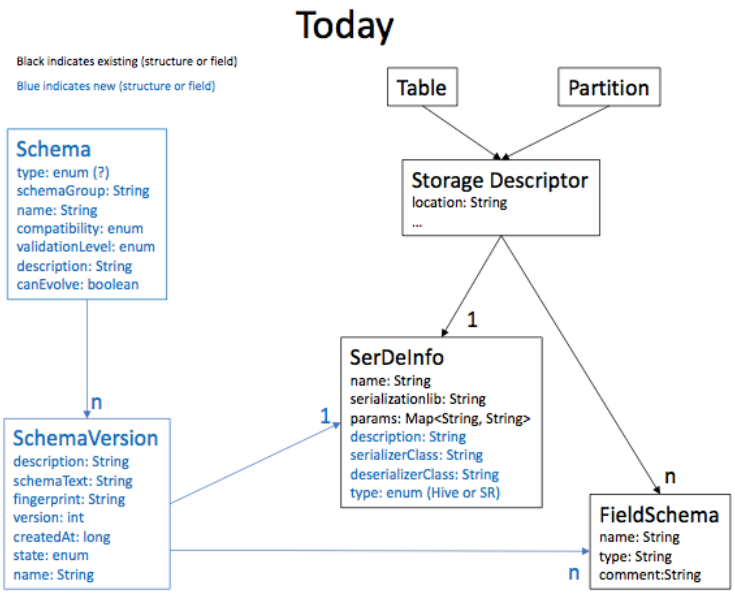
Type mapping:

Schema Registry Type	**Metastore Type**
boolean	boolean
byte	tiny int
short	small int
integer	integer
long	big int
float	float
double	double
string	string
binary	binary
nested	Struct
array	list
blob	binary

Adding additional structures to the Metastore for SchemaRegistry will be a multi-step process. Once schemas are abstracted out as a separate concept it makes sense for Hive to make use of this as well, since it will reduce the number of schemas Hive has to store. However, the change will be disruptive for Hive and will take significant effort to do in a backwards compatible way. Thus it will not be done as part of the initial migration.

The following diagrams layout the changes to the Thrift objects. This will flow through to changes in the database.

The initial changes look like:

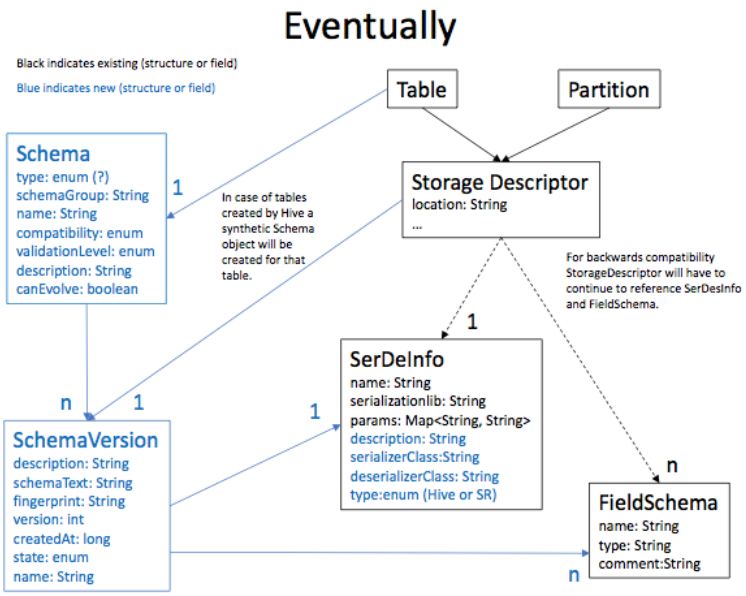


`SerDesInfo` will need additional fields added, since SR supports additional fields that Hive does not. Most of these are self explanatory. The `deserializerClass` and `serializerClass` fields are added because SR assumes the serializer and deserializer can be separate classes, while Hive assumes that one class handles both. The `type` field is added so that callers can discriminate Hive SerDes from SR SerDes, with the current valid values being `Hive` and `SchemaRegistry`.

A new object type `ISchema` will be added to represent an SR schema. `SchemaVersion` will be added to represent a version of a schema in SR. (It is called `ISchema` rather than `Schema` because there is already a Schema object in Thrift.)

Currently SR has a one-to-many mapping between schema and SerDes. This needs to be changed to a one-to-one relationship with SchemaVersion so that users understand which SerDe to use to decode their data. I have discussed this with the Schema Registry team and they agree.

This version does not have a way to specify the location of data. This fits with current SR functionality. In the future SR would like to include location as part of the schema definition. Also, this will be required to allow Hive to access any data SR data. To address these issues and allow Hive to benefit from the advantages of having schema as an independent object in the Metastore (noted above), the eventual plan is



In this plan a Table will have a relationship with a Schema. In the case of SQL engines like Hive, when a table is created, a schema will be created to go with it (assuming the user is not casting a table over an existing schema, which will also be possible). A StorageDescriptor will then reference a SchemaVersion. For backwards compatibility the StorageDescriptor will have to keep references to SerDes and FieldSchema, even though the proper access to this information will now be via SchemaVersion. This is required for external clients who may not have updated code, for older versions of Hive, and because those references are required fields in the Thrift object and thus cannot be null. Checks will need to be put into the Metastore that when new StorageDescriptors are added they properly reference a SchemaVersion or in a backwards compatible writing mode, create a Schema and SchemaVersion to be referenced.