Moving Beyond Moving Bytes

Joey Frazee

Suneel Marthi

September 12, 2017

Flink Forward, Berlin, Germany

$WhoAreWe

Joey Frazee

@jfrazee

- Product Solutions Architect, Hortonworks
- Committer on Apache NiFi, and PMC on Apache Streams

Suneel Marthi

@suneelmarthi

- Principal Software Engineer, Office of Technology, Red Hat
- Member of Apache Software Foundation
- Committer and PMC on Apache Mahout, Apache OpenNLP, Apache Streams

Agenda

- What is a Schema Registry?
- Why should you Care?
- What Exists Today?
- Different Wire Formats
- Using a Schema Registry
- Using a Schema Registry across a Data pipeline
- Implementation with Flink Deserialization Schemas
- Demo

What is a Schema Registry?

What is a Schema?

- information about a record
- field names, field types, default values and aliases

A schema registry is:

- a centralized, versioned schema repository service
- that supports de-centralized schema-based serialization and deserialization

Why should you care?

Because, Real-time stream processing mandates that you know the semantics of your data:

- Interesting operations on streaming
  - joins, projection, aggregation, filtering, streaming SQL

  all require prior knowledge of the types and values of data
- Otherwise, you're just moving bytes and counting anonymous things (you don't need something as powerful as Flink to do that)

And, using embedded schemas is an (unnecessary) overhead:

- The schema can be larger than data
- And it introduces a copy of the schema for every message or topic

And, including schemas in your project is bad:

- Is not recommended b/c it tightly couples the project to your data
- Creates practical scalability issues across system and application boundaries

In general a schema registry offers or implements :

- Schema database
- Version control strategy
- Application API for serialization and deserialization according to the schema
- API service (e.g., REST) for schema management
- Way to acquire, include, or pull in binary artifacts (e.g., SerDes) from the service
- Wire format that encodes a schema *identifier* along with contents in serialized objects

What Exists Today

3 Options

Cask Schema Registry

- A schema serving layer for Avro and Protobuf to support data preparation and validation in Cask CDAP Wrangler

Confluent Schema Registry

- An interface for storing and retrieving Avro schemas for efficient serialization in Kafka and interop with Kafka Streams

Hortonworks Registry

- Shared repository of schemas and SerDes to support Avro schema sharing, record processing and serialization in and across applications (e.g., Apache NiFi, Hortonworks Streaming Analytics Manager)

Wire Formats

Cask (N/A?)

Confluent (5 byte header)

- Magic byte/protocol version (byte): 0
- Schema id (int): 1-4

Hortonworks (13 byte header)

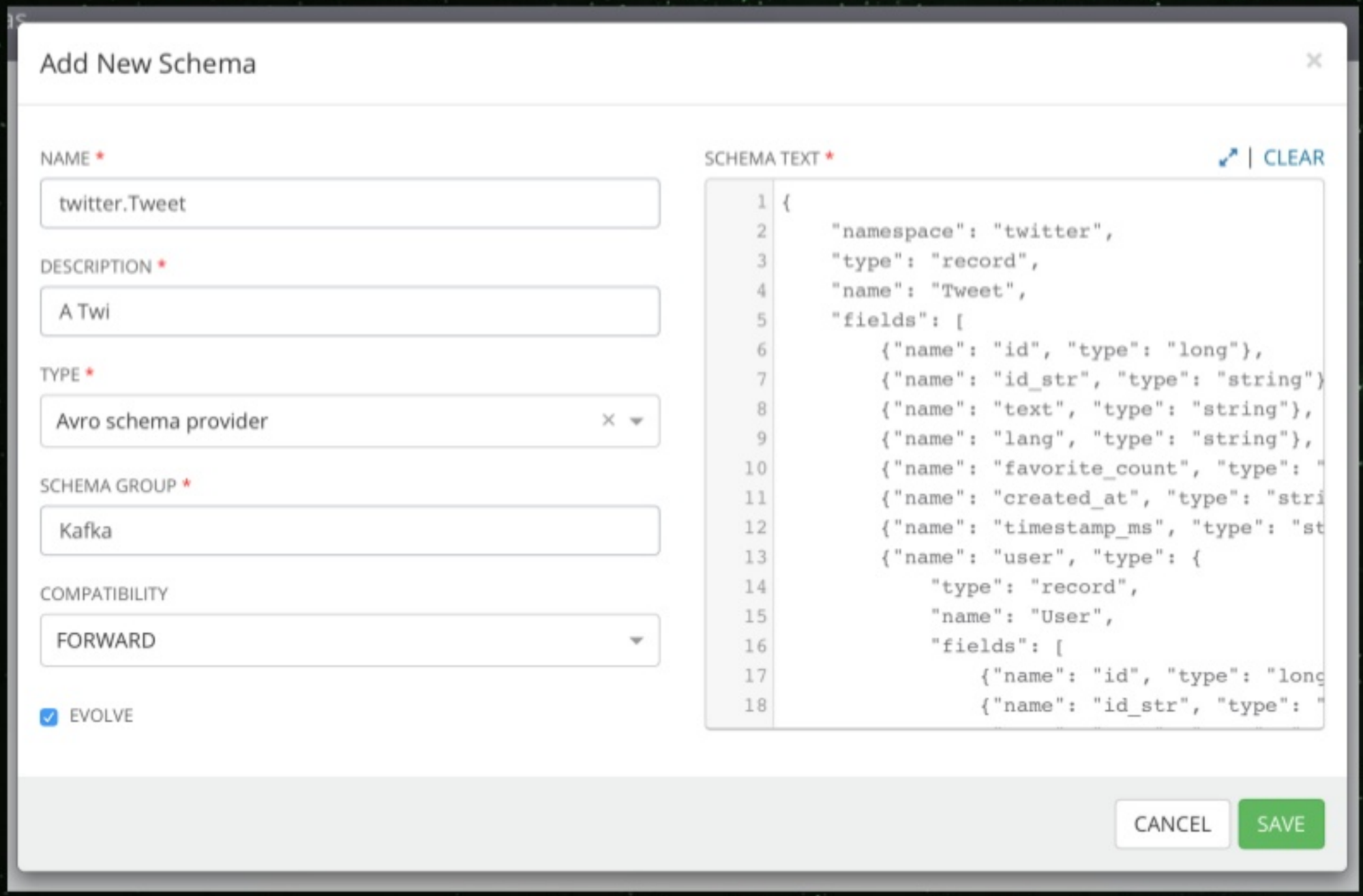
- Magic byte/protocol version (byte): 0
- Schema id (long): 1-8
- Schema version (int): 9-12

Feature Comparison

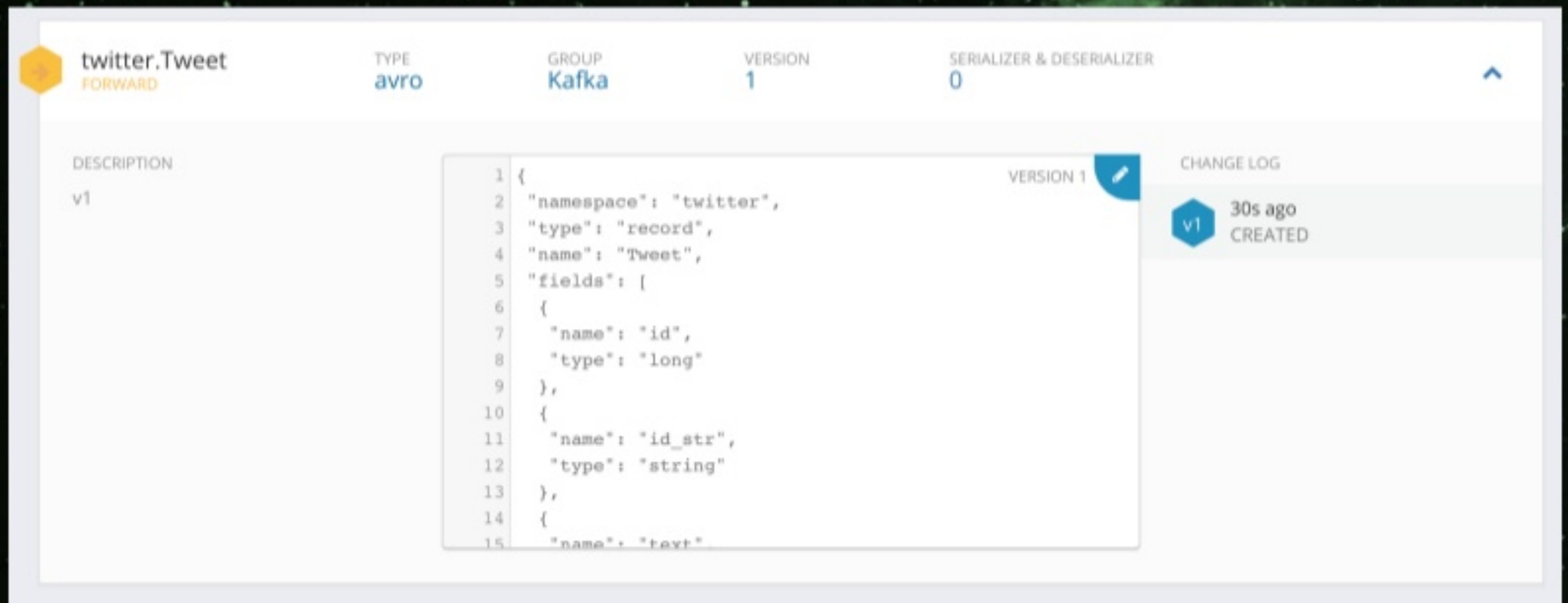
	REST API	Schemas	Custom SerDes	Storage	HA	UI	Maven Plugin	Schema Compatibility Checking	Kafka Integration
Cask	Y	Avro, Protobuf		Cask CDAP DataSet	?	Y			?
Confluent	Y	Avro		Kafka Topic	master/slave		Y	Y	Y
Hortonworks	Y	Avro	Y	JDBC, HDFS	storage + load balancer/proxy				Y

Using a Schema Registry

Add a New Schema



Schema Entry



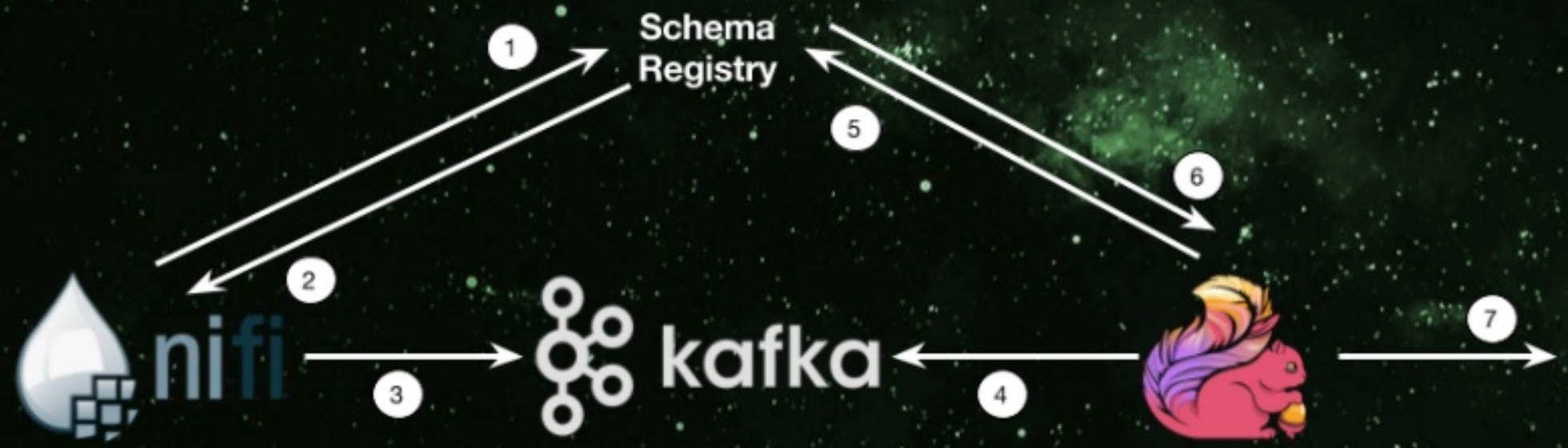
Edit Schema


Edit Version
DESCRIPTION *
Added user.statuses_count
SCHEMA TEXT *
CLEAR
"fields": [
{"name": "text", "type": "string"}
]
}
}},
{"name": "user_mentions", "type": {
"type": "array", "items": {
"type": "record",
"name": "UserMention",
"fields": [
{"name": "id", "type": "long"},
{"name": "id_str", "type": "strin
{"name": "screen_name", "type": "
{"name": "name", "type": "string"}
]
}
}}
]
}}
]
}
CANCEL
SAVE

Schema Version

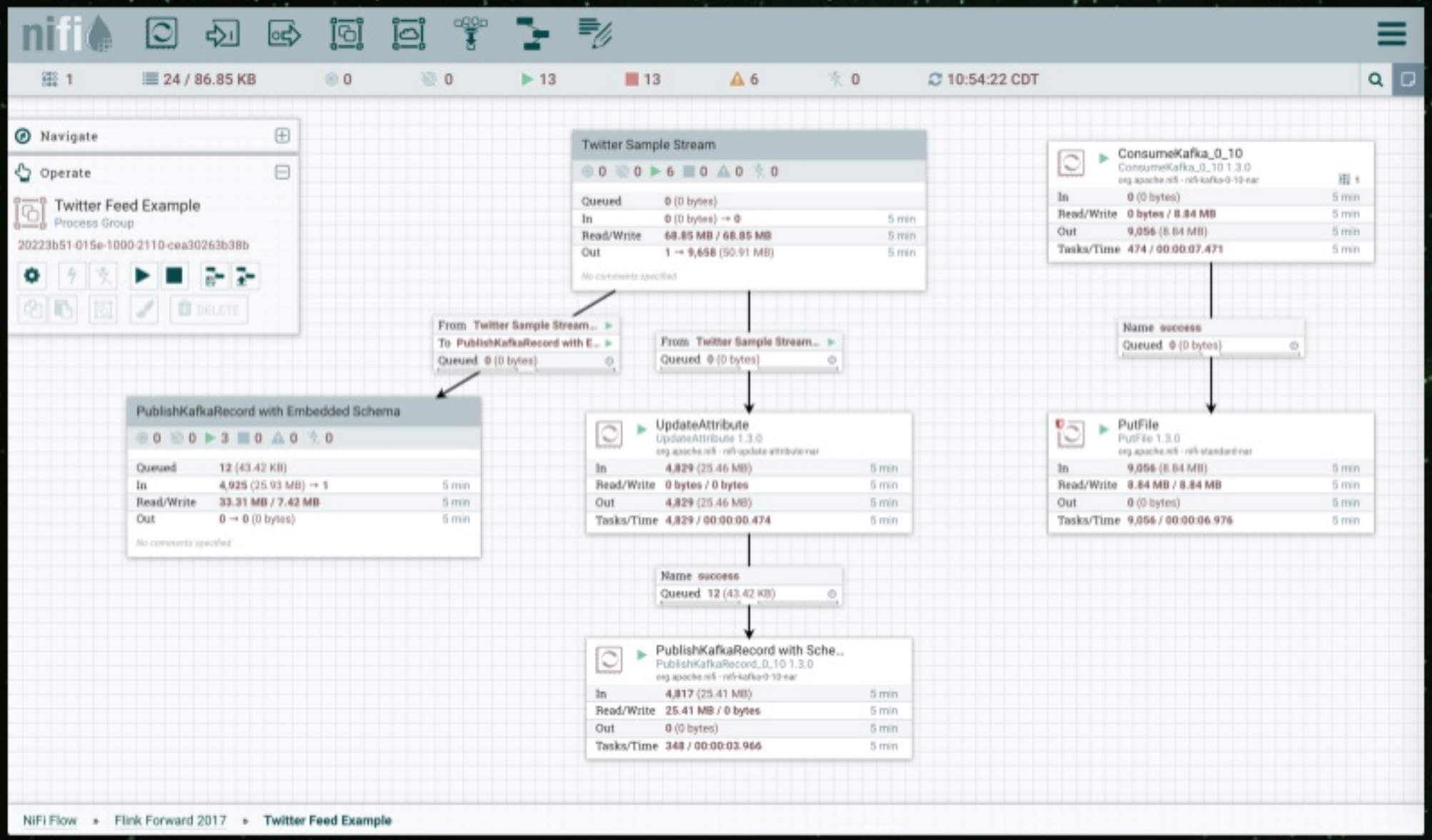
Using a Schema Registry across a Data pipeline

Example Data Pipeline



1. Request schema from schema registry service via schema id
2. Receive the associated schema
3. Serialize the message contents according to the schema, packed with the encoded schema metadata, and publish to Kafka
4. Consume from Kafka and decode the message into its schema metadata and contents
5. Request the schema from schema registry service via schema id
6. Receive the associated schema
7. Deserialize the contents according to the schema and do cool stuff

Apache NiFi Twitter Feed Example



Schema Access Strategies

Embedded schema:

- Whole schema is written out with the message contents (in Avro this corresponds to DataFileReader/Writer)

Schema metadata reference:

- Schema id and other metadata are written as a header with the contents

Implicit schema:

- No schema is presented and application must know what it's expecting or iterate through the universe of possibilities

Serialization with Embedded Schema

```
0000000   O   b   j 001 002 026   a   v   r   o   .   s   c   h   e   m
0000020   a 232 022   {   "   t   y   p   e   "   :   "   r   e   c   o
0000040   r   d   "   ,   "   n   a   m   e   "   :   "   T   w   e   e
0000060   t   "   ,   "   n   a   m   e   s   p   a   c   e   "   :   "
0000100   t   w   i   t   t   e   r   "   ,   "   f   i   e   l   d   s
0000120   "   :   [   {   "   n   a   m   e   "   :   "   i   d   "   ,
0000140   "   t   y   p   e   "   :   "   l   o   n   g   "   }   ,   {
0000160   "   n   a   m   e   "   :   "   i   d   _   s   t   r   "   ,
0000200   "   t   y   p   e   "   :   "   s   t   r   i   n   g   "   }
0000220   ,   {   "   n   a   m   e   "   :   "   t   e   x   t   "   ,
0000240   "   t   y   p   e   "   :   "   s   t   r   i   n   g   "   }
0000260   ,   {   "   n   a   m   e   "   :   "   l   a   n   g   "   ,
0000300   "   t   y   p   e   "   :   "   s   t   r   i   n   g   "   }
0000320   ,   {   "   n   a   m   e   "   :   "   f   a   v   o   r   i
0000340   t   e   _   c   o   u   n   t   "   ,   "   t   y   p   e   "
0000360   :   "   l   o   n   g   "   }   ,   {   "   n   a   m   e   "
0000400   ...
```

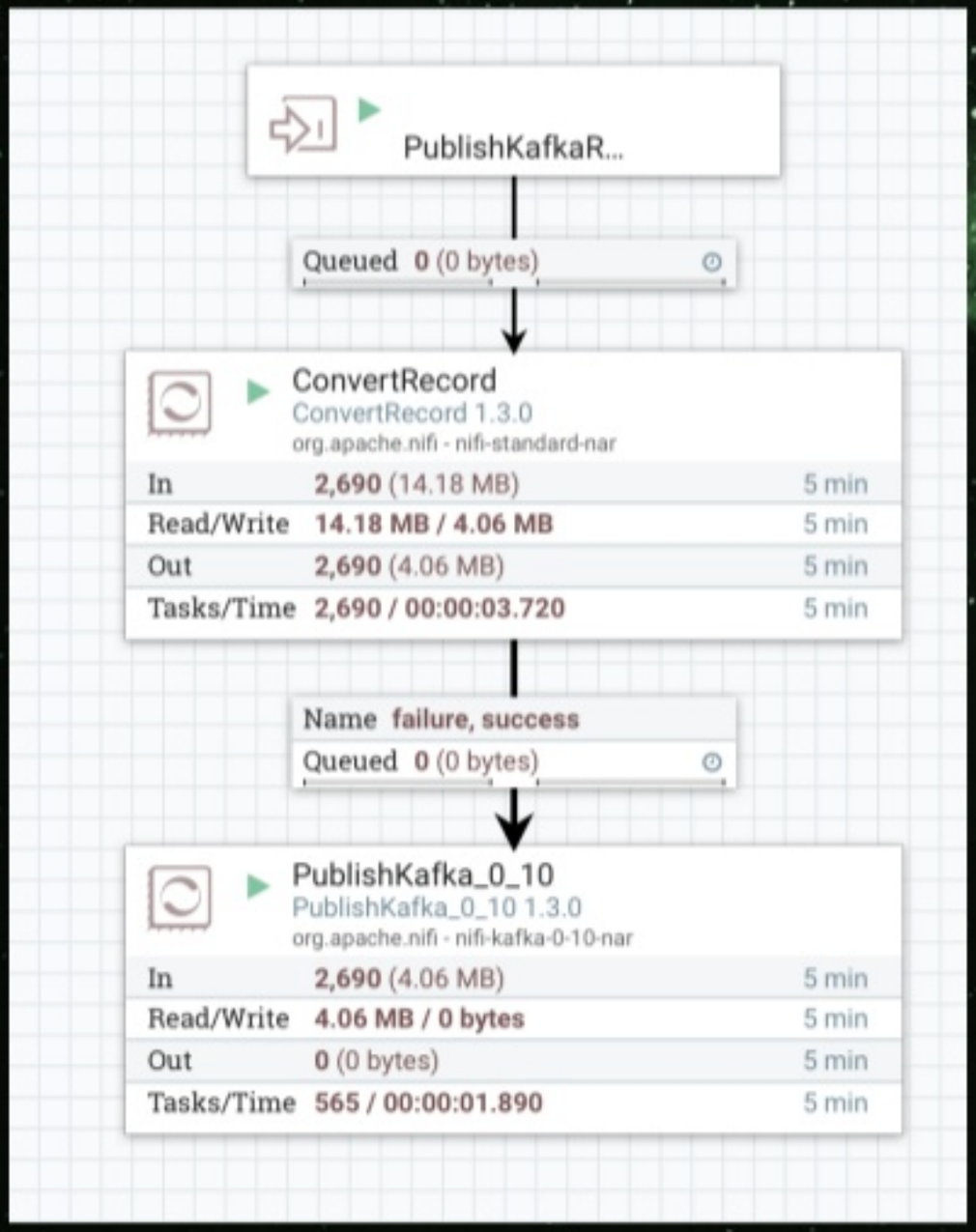
Serialization with Hortonworks Schema Reference

```
0000000 001  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0 001 200 200
0000020   ? 214 204   ? 227 031   $   9   0   7   3   1   2   6   6
0000040   5   8   8   6   8   1   7   2   8   j   R   T       @   B
0000060   S   _   t   w   t   :       T   h   a   n   k       y   o
0000100       ? 230 201   ?   ? 217       h   t   t   p   s   :   /
0000120   t   .   c   o   /   8   g   w   a   z   v   b   U   J   C 004
0000140   e   n  \0   <   M   o   n       S   e   p       1   1
0000160   8   :   3   9   :   3   1       +   0   0   0   0       2
0000200   1   7 032   1   5   0   5   1   5   5   1   7   1   6   6
0000220 226   ? 225 221  \b 024   1   0   9   1   7   4   6   6   9
0000240 006   P   M   Y 022   A   n   a   t   i   A   m   i   r 002 032
0000260   M   a   t   o   k   i       P   l   a   n   e   t  \0 002
0000300   D   o   n   t       b   l   a   m   e       m   e   ,   I
0000320   m       w   e   i   r   d   ?   ?  \n   ?  \f   ? 031  \0 001
0000340   N 214   ?   ?   ? 002 022   3   3   5   1   4   1   6   3
0000360 016   B   T   S   _   t   w   t 036   ?   ?   ?   ? 203 204
0000400 206 214   ? 205 204   ? 213   ?  \0
```

Serialization with Confluent Schema Reference

```
0000000   \0   \0   \0   \0    Q 230    ?    ?    ? 201    ?    ? 227 031    $    9
0000020    0    7    3    1    8    0    9    5    0    1    7    9    9    6    2    8
0000040    8    v    @    _    _    k    i    l    e    y         @    o    n    l    y
0000060    s    i    n    w    o    r    l    d         Y    e    a    h         w    t
0000100    f    ?    ?    ?         T    h    a    t         i    s              s    u    p
0000120    e    r         w    e    i    r    d              ? 237 230    ? 004    e    r
0000140   \0    <    M    o    n         S    e    p         1    1         1    9    :
0000160    0    1    :    0    5         +    0    0    0    0         2    0    1    7
0000200  032    1    5    0    5    1    5    6    4    6    5    6    6    4    ?    ?
0000220    ? 211 216    ? 204    ? 023    $    7    0    2    0    0    8    8    6
0000240    7    3    6    6    9    3    2    4    8    1   \n    c    y    n    d    i
0000260  030    c    y    n    d    a    q    u    i    l    l    l    l 022    S    a
0000300    n    d    y    ,         U    T   \0    >    a              y    o    u    n    g
0000320         m    o    m         l    i    v    i    n    g              i    n         s
0000340    u    b    u    r    b    i    a    .   \0    ?   \a 204   \a
```

Convert Record Processor Group



ConvertRecord Properties

Processor Details

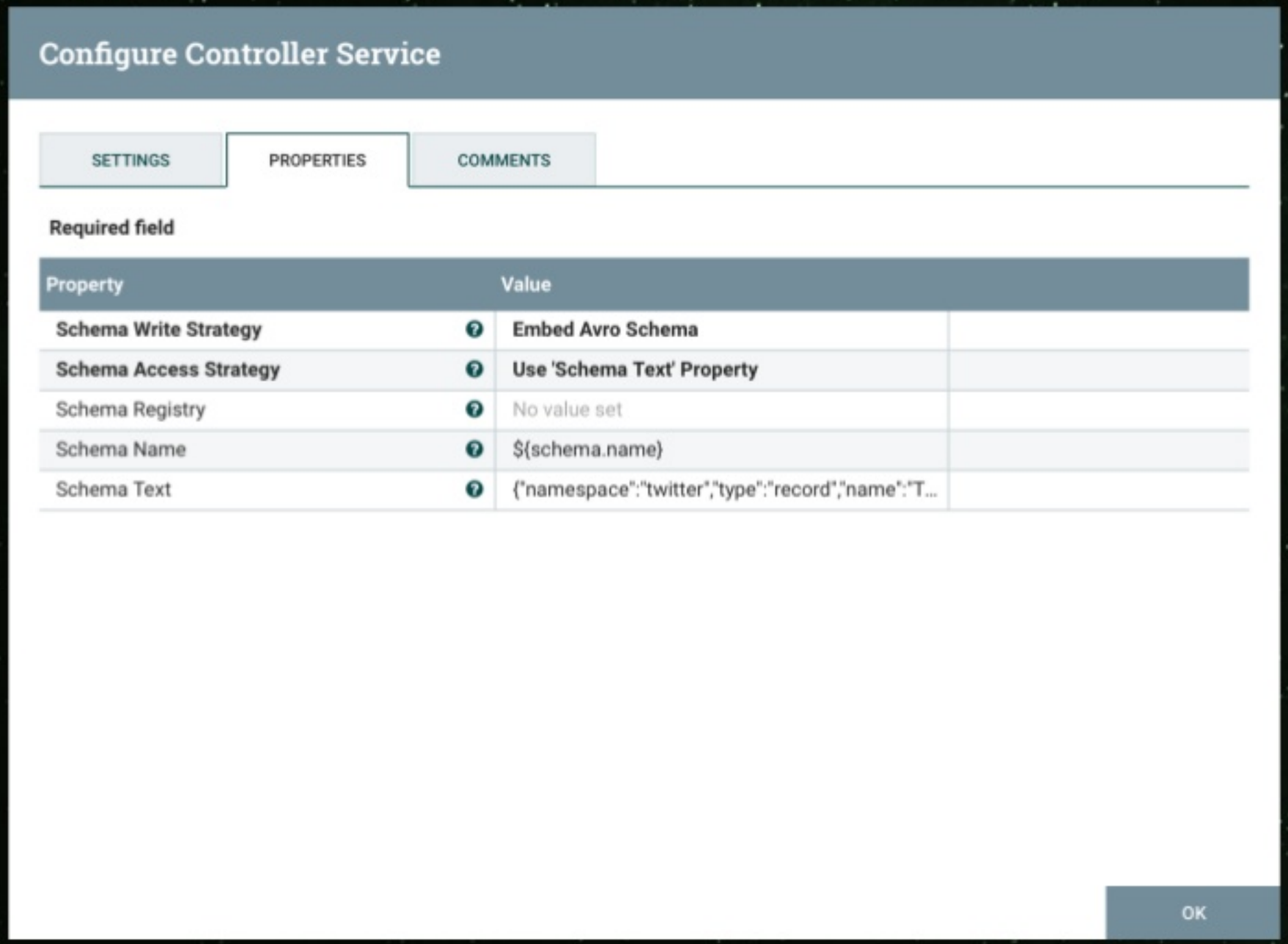
SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

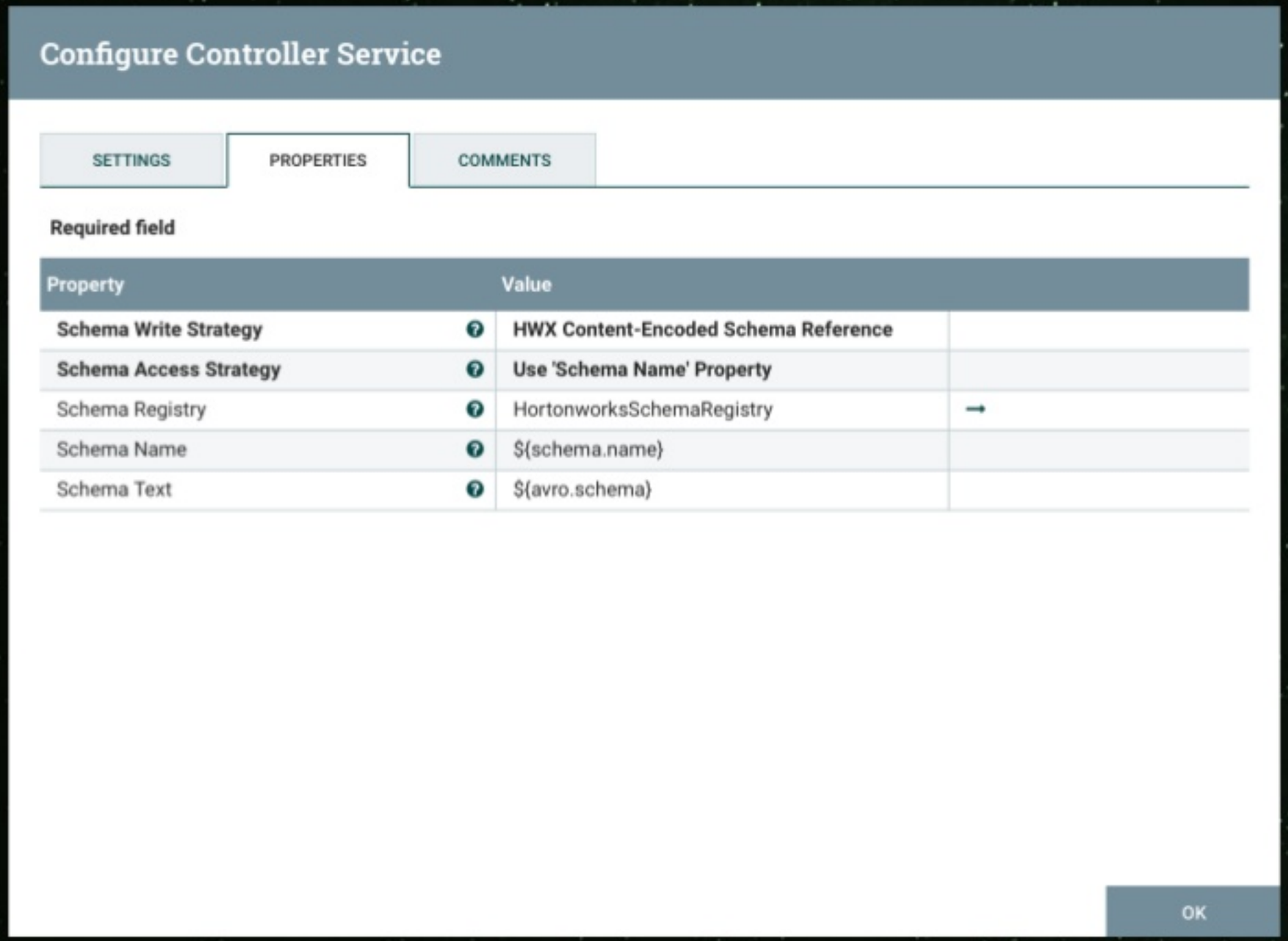
Property	Value
Record Reader	Tweet JsonTreeReader with Schema Text
Record Writer	Tweet AvroRecordSetWriter with Schema Text

OK

AvroRecordSetWriter Properties without Schema Registry



AvroRecordSetWriter Properties with Schema Registry



PublishKafkaRecord Properties

Processor Details

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value	
Kafka Brokers	**localhost:9092**	
Topic Name	**tweets.avro.encoded**	
Record Reader	**Tweet JsonTreeReader with HWX Schema Registry**	→
Record Writer	**Tweet AvroRecordSetWriter with HWX Schema Registry**	→
Security Protocol	**PLAINTEXT**	
Kerberos Service Name	No value set	
Kerberos Principal	No value set	
Kerberos Keytab	No value set	
SSL Context Service	No value set	
Delivery Guarantee	**Best Effort**	
Message Key Field	No value set	
Max Request Size	**1 MB**	
Acknowledgment Wait Time	**5 secs**	
Max Metadata Wait Time	**5 sec**	

OK

Implementation with Flink Deserialization Schemas

Hortonworks Deserialization Schema

```scala

@SerialVersionUID(1L)
class HortonworksRegistryDeserializationSchema[T](avroType: Class[T], url: String = "http://localhost:9090/api/v1") extends DeserializationSchema[T] {

```

```scala

  private[this] def getSchemaInfo(message: Array[Byte]): (Long, Int) = {
    if (message.length < 13)
      throw new IllegalArgumentException(s"Message is too short for schema
        encoding reference: ${message.mkString(" ")}")

    val buffer = ByteBuffer.wrap(message)

    val encodingVersion = buffer.get
    if (encodingVersion > 1)
      throw new IllegalArgumentException(s"Unrecognized schema encoding
        version: ${encodingVersion}")

    val schemaId = buffer.getLong
    val schemaVersion = buffer.getInt

    (schemaId, schemaVersion)
  }

  private[this] def getContentsWithInfo(message: Array[Byte]): (Array[Byte],
    Long, Int) = {
    val (schemaId, schemaVersion) = getSchemaInfo(message)
    val contents = message.drop(13)

    if (contents.isEmpty)
      throw new IllegalArgumentException(s"Message is 0 bytes")

    (contents, schemaId, schemaVersion)
  }

```

```scala

  private[this] def getSchema(schemaId: Long, schemaVersion: Int):
    Option[Schema] = {
    for {
      schemaMetadataInfo <-
        Option(registryClient.getSchemaMetadataInfo(schemaId))
      schemaMetadata <- Option(schemaMetadataInfo.getSchemaMetadata())
      schemaName <- Option(schemaMetadata.getName())
      schemaVersionKey = new SchemaVersionKey(schemaName, schemaVersion)
      schemaVersionInfo <-
        Option(registryClient.getSchemaVersionInfo(schemaVersionKey))
      schemaText <- Option(schemaVersionInfo.getSchemaText())
    } yield new Schema.Parser().parse(schemaText)
  }

  override def deserialize(message: Array[Byte]): T = {
    val (contents, schemaId, schemaVersion) = getContentsWithInfo(message)

    getSchema(schemaId, schemaVersion) match {
      case Some(schema) => {
        val reader: DatumReader[T] =
          if (avroType == classOf[GenericRecord])
            new GenericDatumReader[T](schema)
          else if (classOf[SpecificRecordBase].isAssignableFrom(avroType))
            new SpecificDatumReader[T](schema)
          else
            new ReflectDatumReader[T](schema)

        try {
          this.decoder = DecoderFactory.get().binaryDecoder(contents, decoder)
          reader.read(null.asInstanceOf[T], decoder)
        } catch {
          case e: Exception =>
            throw new RuntimeException(e)
        }
      }
      case _ =>
        throw new SchemaNotFoundException(s"No schema found matching id and
          version: ${schemaId}v${schemaVersion}")
    }
  }
```

Confluent Deserialization Schema

```scala

@SerialVersionUID(1L)
class ConfluentRegistryDeserializationSchema[T](avroType: Class[T], url: String = "http://localhost:8081/") extends DeserializationSchema[T] {

```

```scala

  private[this] def getSchemaId(message: Array[Byte]): Int = {
    if (message.length < 5)
      throw new IllegalArgumentException(s"Message is too short for schema
        encoding reference: ${message.mkString(" ")}")

    val buffer = ByteBuffer.wrap(message)

    val magicByte = buffer.get
    if (magicByte > 0)
      throw new IllegalArgumentException(s"Unrecognized magic byte:
        ${magicByte}")

    val schemaId = buffer.getInt

    schemaId
  }

  private[this] def getContentsWithSchemaId(message: Array[Byte]):
    (Array[Byte], Int) = {
    val schemaId = getSchemaId(message)
    val contents = message.drop(5)

    if (contents.isEmpty)
      throw new IllegalArgumentException(s"Message is 0 bytes")

    (contents, schemaId)
  }

```

```scala

  private[this] def getSchema(schemaId: Int): Option[Schema] = {
    for {
      result <- Try(registryClient.getById(schemaId)).toOption
      schema <- Option(result)
    } yield schema
  }

  override def deserialize(message: Array[Byte]): T = {
    val (contents, schemaId) = getContentsWithSchemaId(message)

    getSchema(schemaId) match {
      case Some(schema) => {
        val reader: DatumReader[T] =
          if (avroType == classOf[GenericRecord])
            new GenericDatumReader[T](schema)
          else if (classOf[SpecificRecordBase].isAssignableFrom(avroType))
            new SpecificDatumReader[T](schema)
          else
            new ReflectDatumReader[T](schema)

        try {
          this.decoder = DecoderFactory.get().binaryDecoder(contents, decoder)
          reader.read(null.asInstanceOf[T], decoder)
        } catch {
          case e: Exception =>
            throw new RuntimeException(e)
        }
      }
      case _ =>
        throw Errors.schemaNotFoundException()
    }
  }

```

Next Steps with Apache Flink

Higher level SerDes for:

- Source/Sink
- TableSource/TableSink

References

- Apache NiFi — Records and Schema Registries - https://bryanbende.com/development/2017/06/20/apache-nifi-records-and-schema-registries
- Confluent Schema Registry — https://github.com/confluentinc/schema-registry
- Github — https://github.com/jfrazee/schema-registry-examples
- HortonWorks Schema Registry — http://github.com/hortonworks/registry
- Record-Oriented Data with NiFi — https://blogs.apache.org/nifi/entry/record-oriented-data-with-nifi

Credits

- Bryan Bende — Staff Software Engineer, Hortonworks and PMC on Apache NiFi
- Bruno P. Kinoshita — PMC on Apache OpenNLP and Apache Commons

Questions ???