Apache NiFi Record Processing

Bryan Bende / @bbende

Staff Software Engineer

September 8th 2017

HORTONWORKS®
POWERING THE FUTURE OF DATA™

Background

- Flow File
  - Unit of work that moves through the data flow
  - Made up of attributes + content
- Attributes are a map of key/value pairs
  - Available in-memory as strings
  - Accessible from expression language
  - Useful for quick decision-making/routing
- Content is arbitrary bytes
  - Flow File is a pointer to the content in the content repository
  - Content is only accessed if the processor needs to operate on it
  - Could pass through many processors without every accessing the content

The Problem

- Specialized processors to operate on different data types
  - SplitJson, EvaluateJsonPath, ConvertJsonToAvro
  - SplitAvro, ExtractAvroMetadata, ConvertAvroToJson
  - SplitText, ExtractText, RouteText
- Sometimes missing conversions
  - No ConvertCsvToJson, so ConvertCsvToAvro then ConvertAvroToJson
- Sometimes missing a specific function for a data type
  - No EvaluateAvroPath, so ConvertAvroToJson then EvaluateJsonPath
- Sometimes implemented with different libraries causing inconsistencies
  - Some Avro processors implemented with Kite, others with Apache Avro libraries
  - Each library may have different features/error-handling

The Solution

- Introduce the concept of a ”record”
- Centralize the logic for reading/writing records into controller services
- Provide standard processors that operate on records
- Can still handle arbitrary data, but process records when appropriate

Record Readers & Writers

- Readers
  - *AvroReader*
  - *CsvReader*
  - *GrokReader*
  - *JsonPathReader*
  - *JsonTreeReader*
  - *ScriptedReader*
- Writers
  - *AvroRecordSetWriter*
  - *CsvRecordSetWriter*
  - *JsonRecordSetWriter*
  - *FreeFormTextRecordSetWriter*
  - *ScriptedRecordSetWriter*

But how is data turned into a record?

- A record has fields, and fields have information like a name and type
- Schemas define the fields of a record and give meaning to the data
- Apache Avro already utilizes schemas, widely used & supported by many tools
- We can use Avro schemas to define a schema for any type of data
- Each reader & writer needs a way to obtain a schema

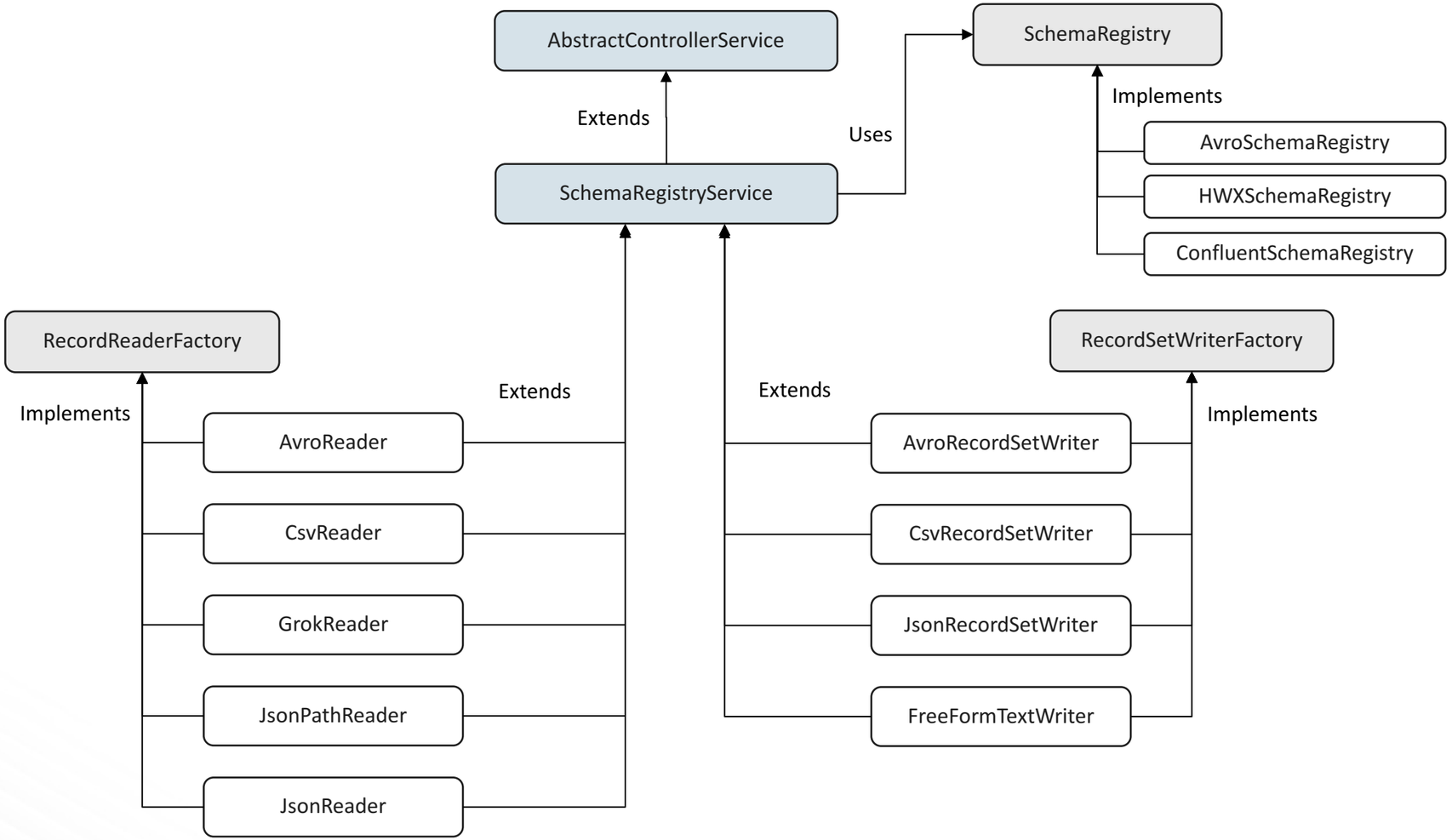
Schema Access Strategy

- *Schema Name*
  - Provide the name of a schema to look up in a Schema Registry, can use EL to obtain the name
- *Schema Text*
  - Provide the text of a schema in reader/writer, can use EL to obtain the text
- *HWX Content-Encoded Schema Reference*
  - Content of the Flow File contains special header referencing a schema in a Schema Registry
- *HWX Schema Reference Attributes*
  - Flow File contains three attributes that will be used to lookup a schema from the configured Schema Registry: *'schema.identifier', 'schema.version',* and *'schema.protocol.version'*
- Readers & writers may have additional options specific to the data type
  - Ex: CsvReader can make a schema on the fly from the column names
  - Ex: AvroReader can use the schema embedded in the Avro data file

Schema Registries

- Avro Schema Registry
  - Access schema by name
  - Only accessible with in NiFi
- Hortonworks Schema Registry
  - Access schema by name and/or version
  - Accessible across systems in the enterprise
  - https://github.com/hortonworks/registry
- Confluent Schema Registry
  - Access schema by name and/or version
  - Accessible across systems in the enterprise
  - https://github.com/confluentinc/schema-registry
  - *Not in an official Apache NiFi release yet, available in master branch (1.4.0-snapshot)*

Full Picture



Record Path

- Domain specific language (DSL) for specifying/accessing fields of a record
- Similar to JSON Path or XPath
- Examples:
  - Child: `/details/address/zip`
  - Descendant: `//zip`
  - Arrays: `/addresses[1]`
  - Maps: `/details/address['zip']`
  - Predicates: `/*[./state != 'NY']`
- More info…
  - https://nifi.apache.org/docs/nifi-docs/html/record-path-guide.html

Record Processors

- Many processors for operating on records
  - *ConvertRecord*
  - *LookupRecord*
  - *PartitionRecord*
  - *QueryRecord*
  - *SplitRecord*
  - *UpdateRecord*
  - *ConsumeKafkaRecord_0_10*
  - *PublishKafkaRecord_0_10*
- Goal is to keep many records per flow file and avoid splitting if possible
- Check latest docs usage details and other record processors
  - https://nifi.apache.org/docs.html

Example – CSV to JSON w/Local Schema Registry

Example - CSV to JSON

- Incoming CSV that looks like:

```
first_name, last_name
John, Smith
Mike, Jones
```

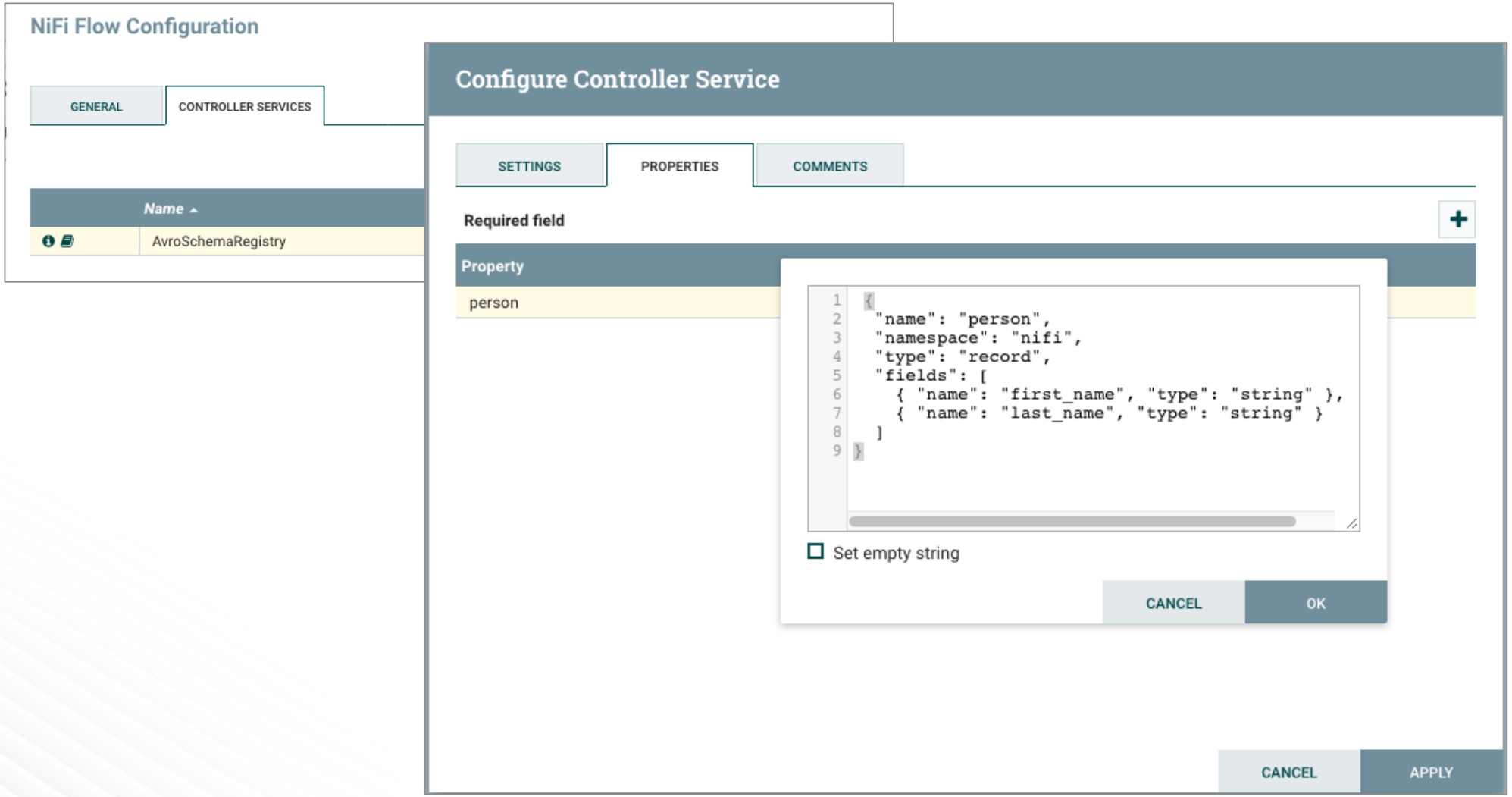
- Want JSON that looks like:

```
[
{“first_name” : “John”, ”last_name” : “Smith”},
{“first_name” : “Mike”, “last_name” : “Jones”}
]
```

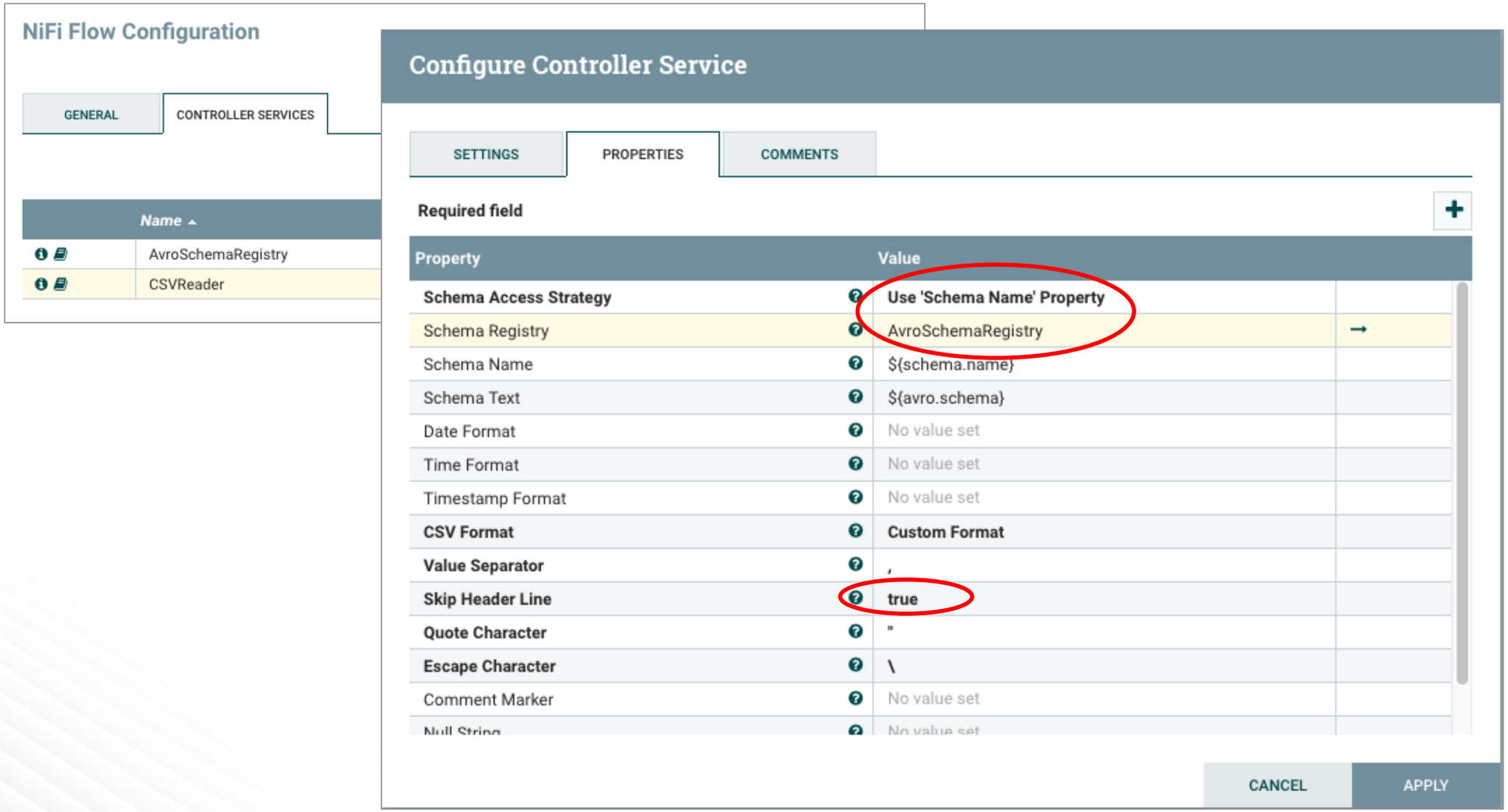
Step 1 – Define an Avro Schema

```
{
  "name": "person",
  "namespace": "nifi",
  "type": "record",
  "fields": [
    { "name": "first_name", "type": "string" },
    { "name": "last_name", "type": "string" }
  ]
}
```

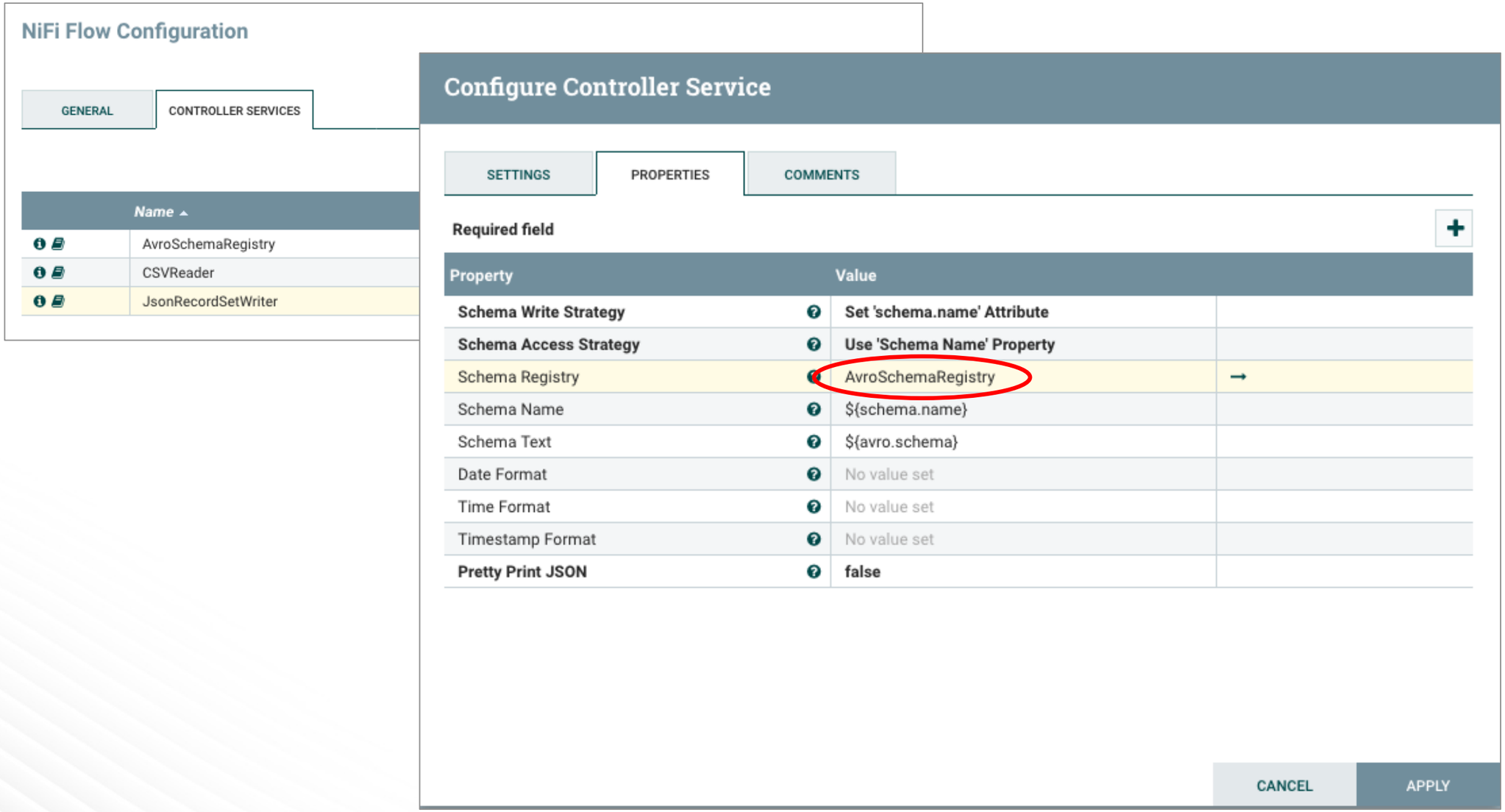
Step 2 - Create a Local Schema Registry & Add Schema



Step 3 - Create a CsvReader

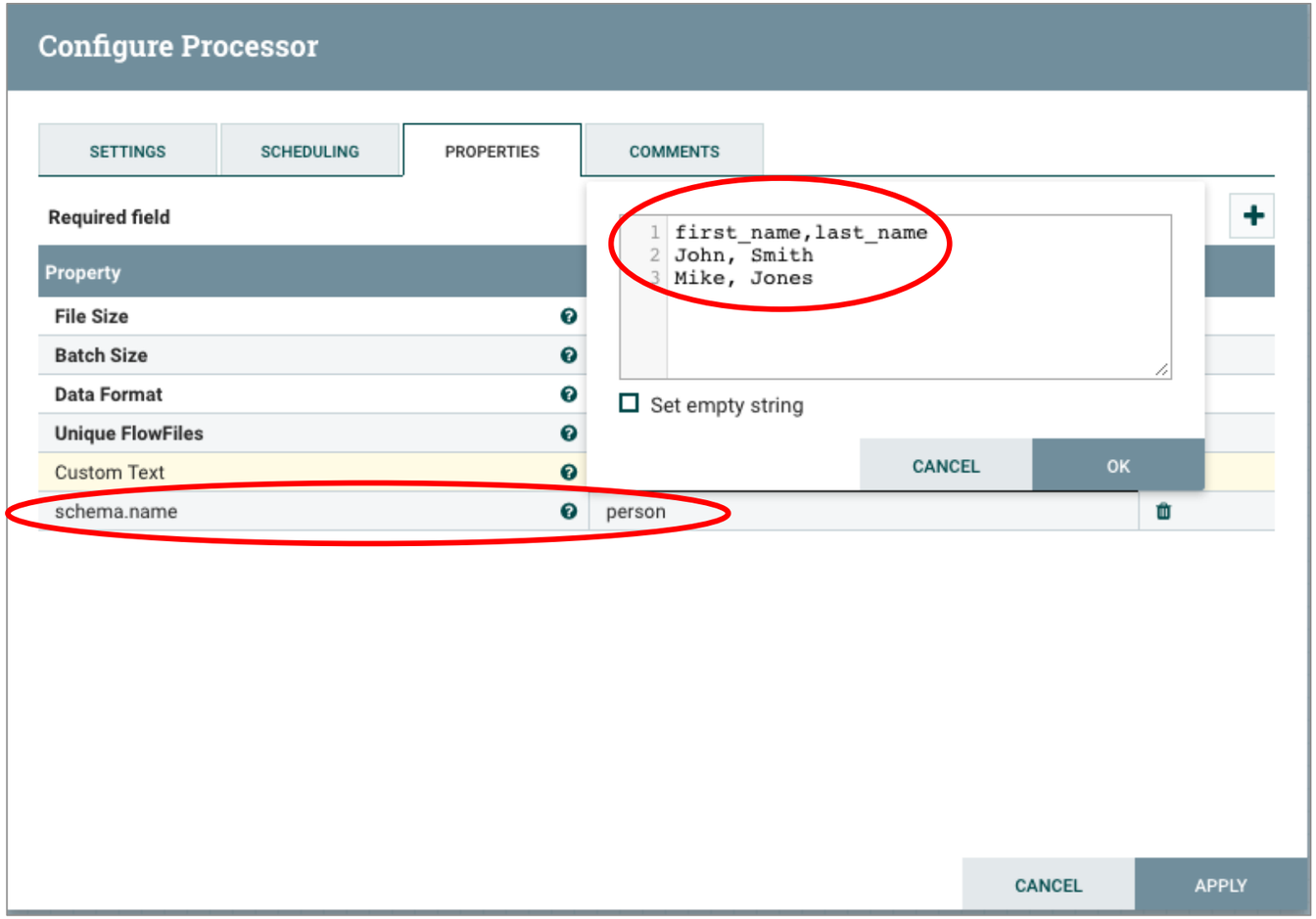


Step 4 – Create a JsonRecordSetWriter



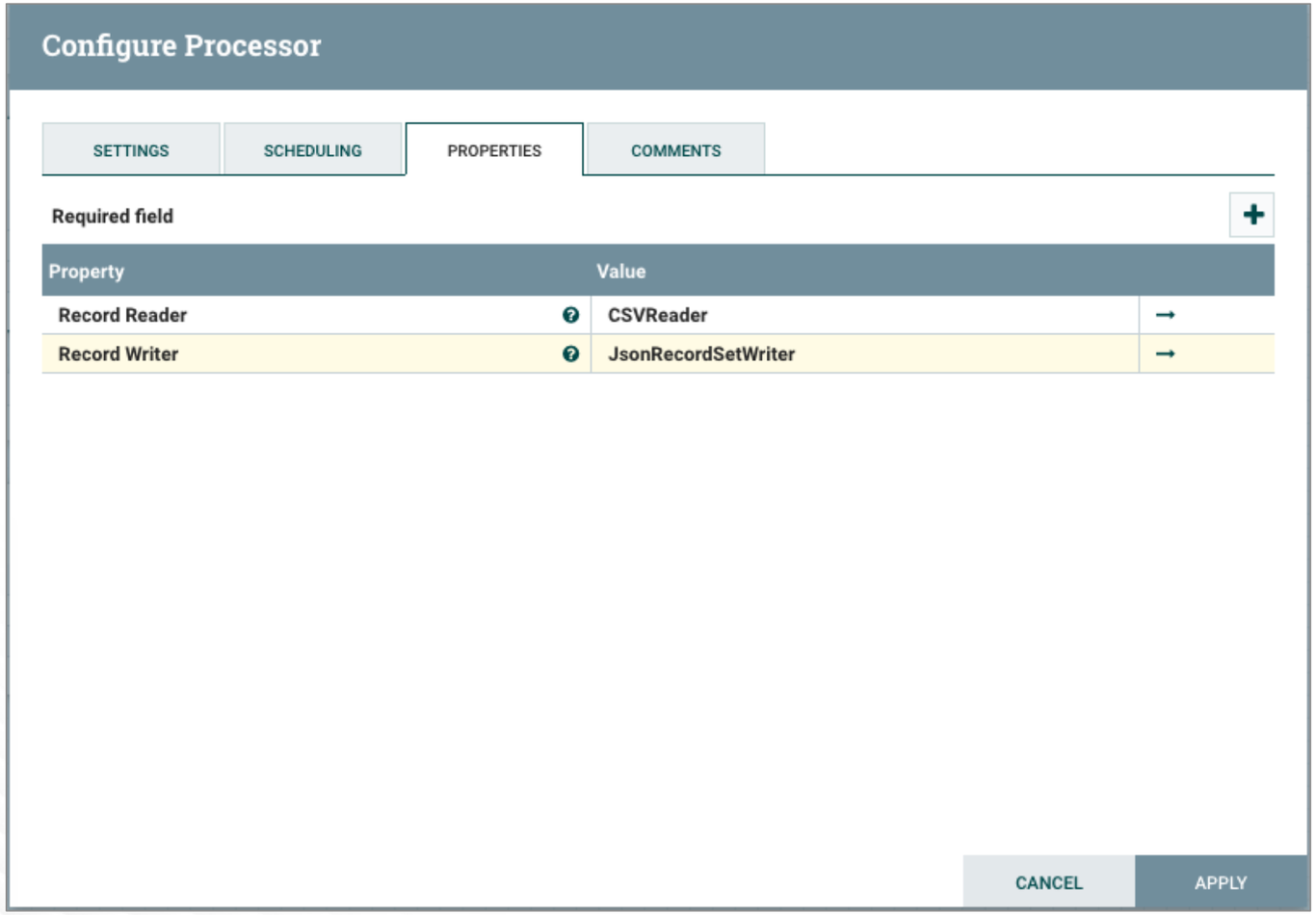
Step 5 – GenerateFlowFile Processor

- Set Run Schedule to something like 10 seconds
- Put example CSV data in Custom Text property
- The reader & writer had their 'Schema Name' set to ${schema.name}
- Add an property called 'schema.name' with the value of 'person' since this is the name in the schema registry



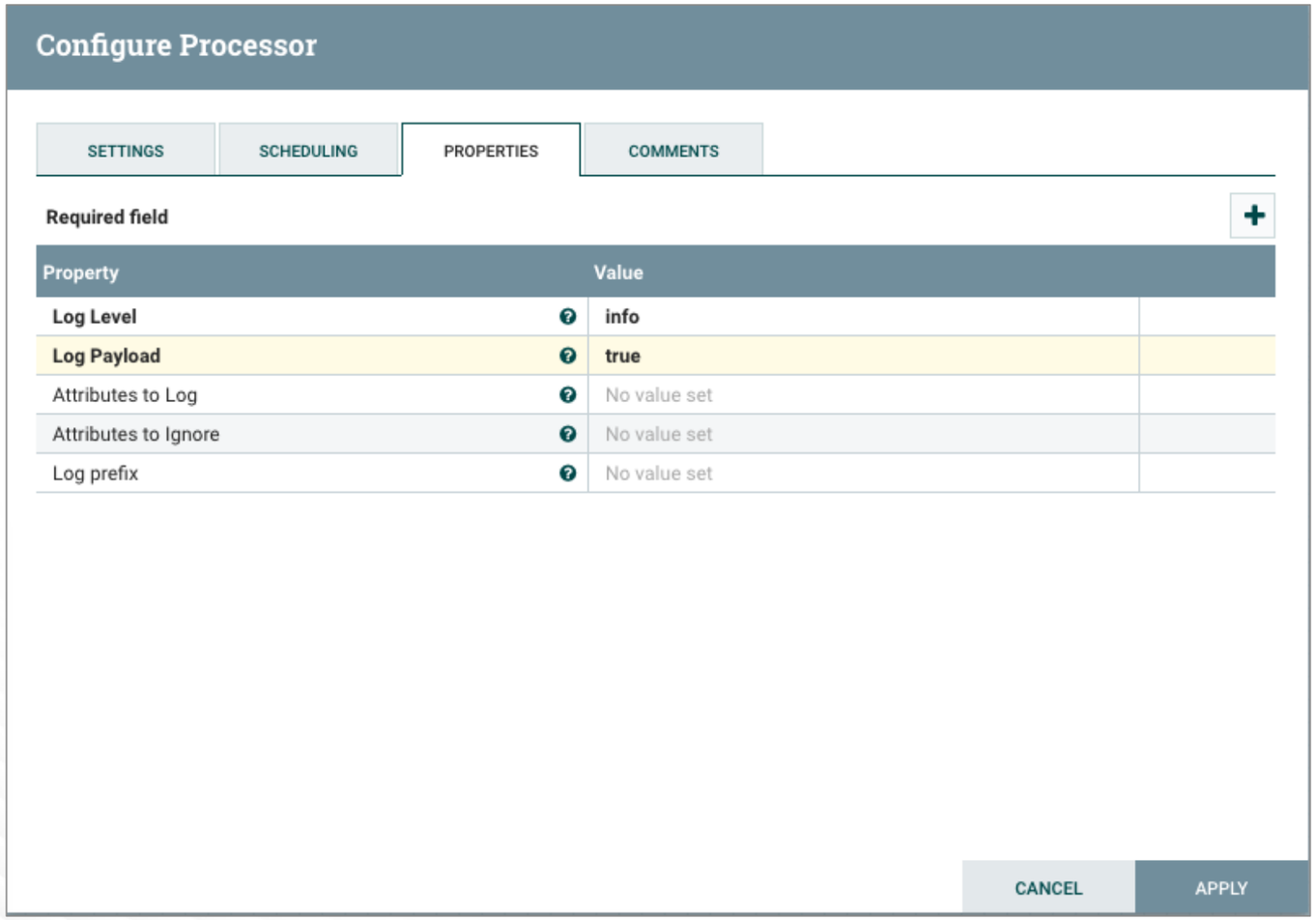
Step 6 – Convert Record Processor

- Select the appropriate reader and writer

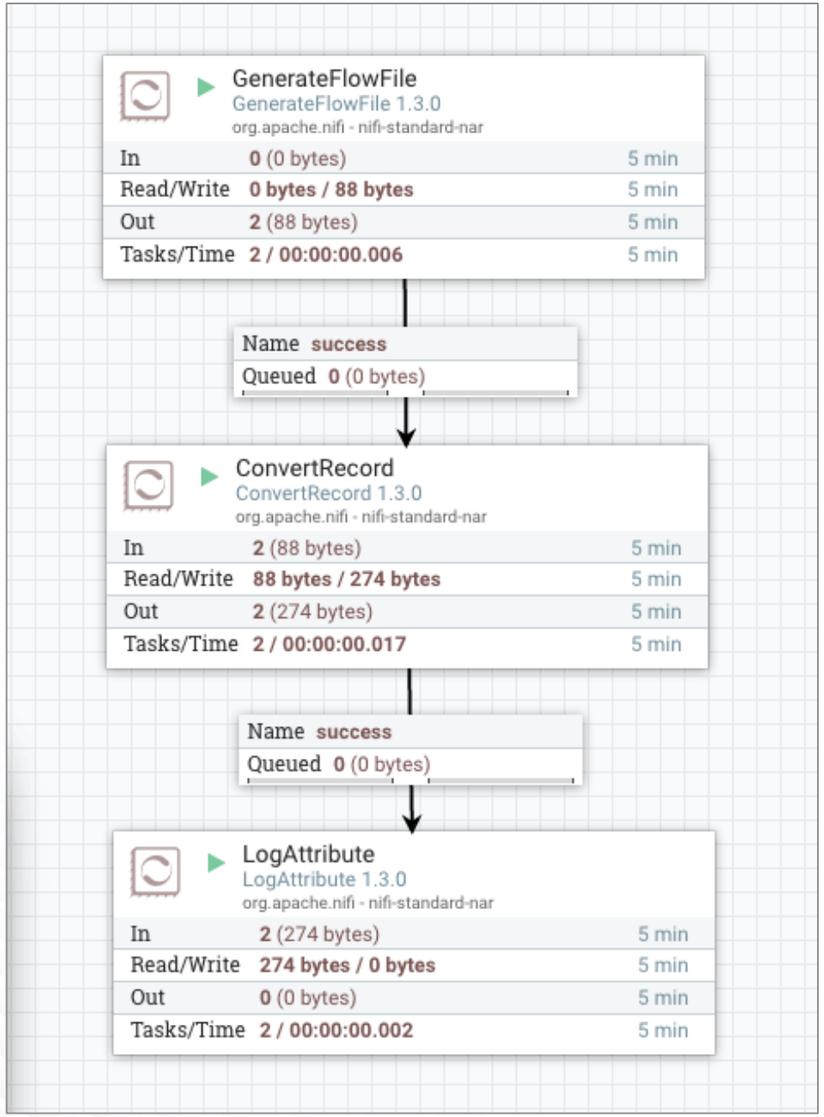


Step 7 - LogAttribute

- Set Log Payload to true



Step 8 – Connect Processors & Run Flow



Step 9 – Check nifi-app.log for JSON

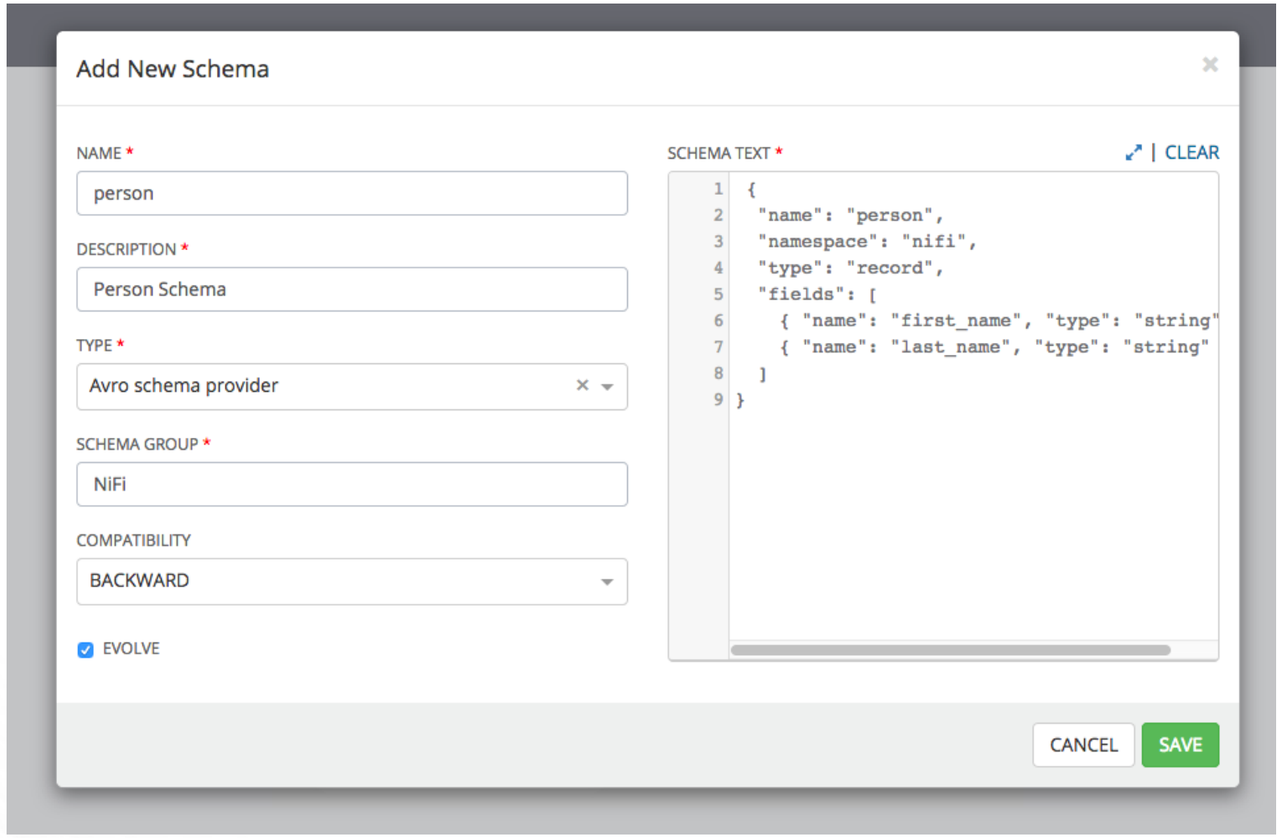
```
--------------------------------------------------Standard FlowFile Attributes
Key: 'entryDate '        Value: 'Thu Aug 31 13:28:02 EDT 2017’
Key: 'lineageStartDate'       Value: 'Thu Aug 31 13:28:02 EDT 2017’
Key: 'fileSize'      Value: '137’
FlowFile Attribute Map Content
Key: 'filename'   Value: '326844487150210’
Key: 'mime.type'        Value: 'application/json’
Key: 'path' Value: './’
Key: 'record.count'     Value: ’2’
Key: 'schema.name'   Value: 'person’
Key: 'uuid' Value: 'e9198166-0cff-400b-a39d-9c8c9c565f85’
--------------------------------------------------
[
{"first_name":"John","last_name":"Smith"},
{"first_name":"Mike","last_name":"Jones"}
]
```

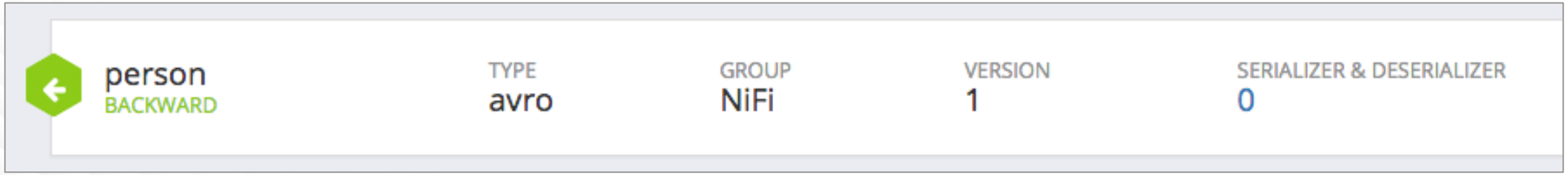
Example – CSV to JSON w/Hortonworks Schema Registry

Step 1 – Run the Hortonworks Schema Registry

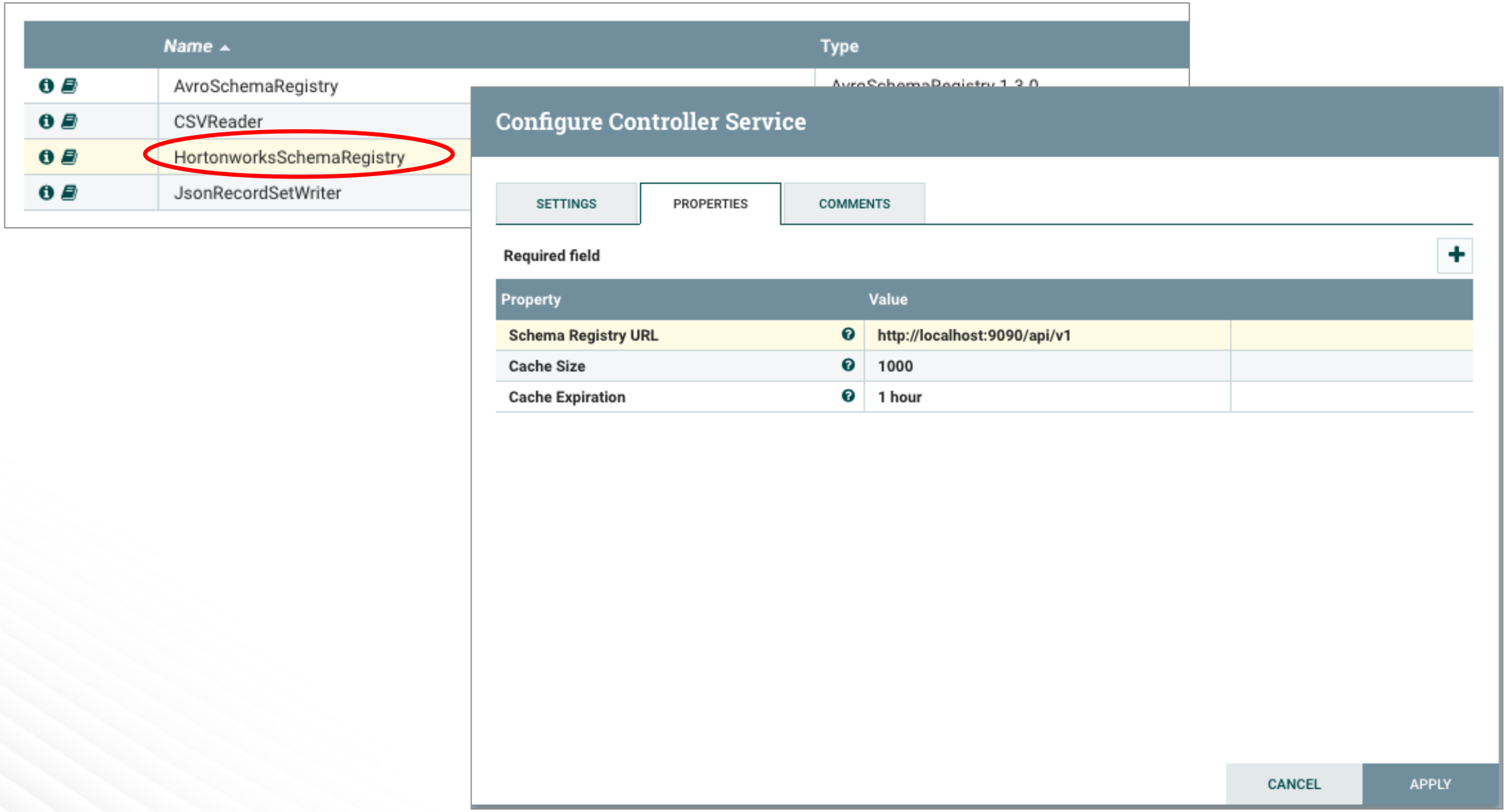
- Download the latest release
  - https://github.com/hortonworks/registry/releases/download/v0.2.1/hortonworks-registry-0.2.1.tar.gz
- Extract the tar and run the application
  - `tar xzvf hortonworks-registry-0.2.1.tar.gz`
  - `cd hortonworks-registry-0.2.1`
  - `./bin/registry-server-start.sh conf/registry-dev.yaml`
- Navigate to registry UI in your browser
  - http://localhost:9090

Step 2 – Add Schema

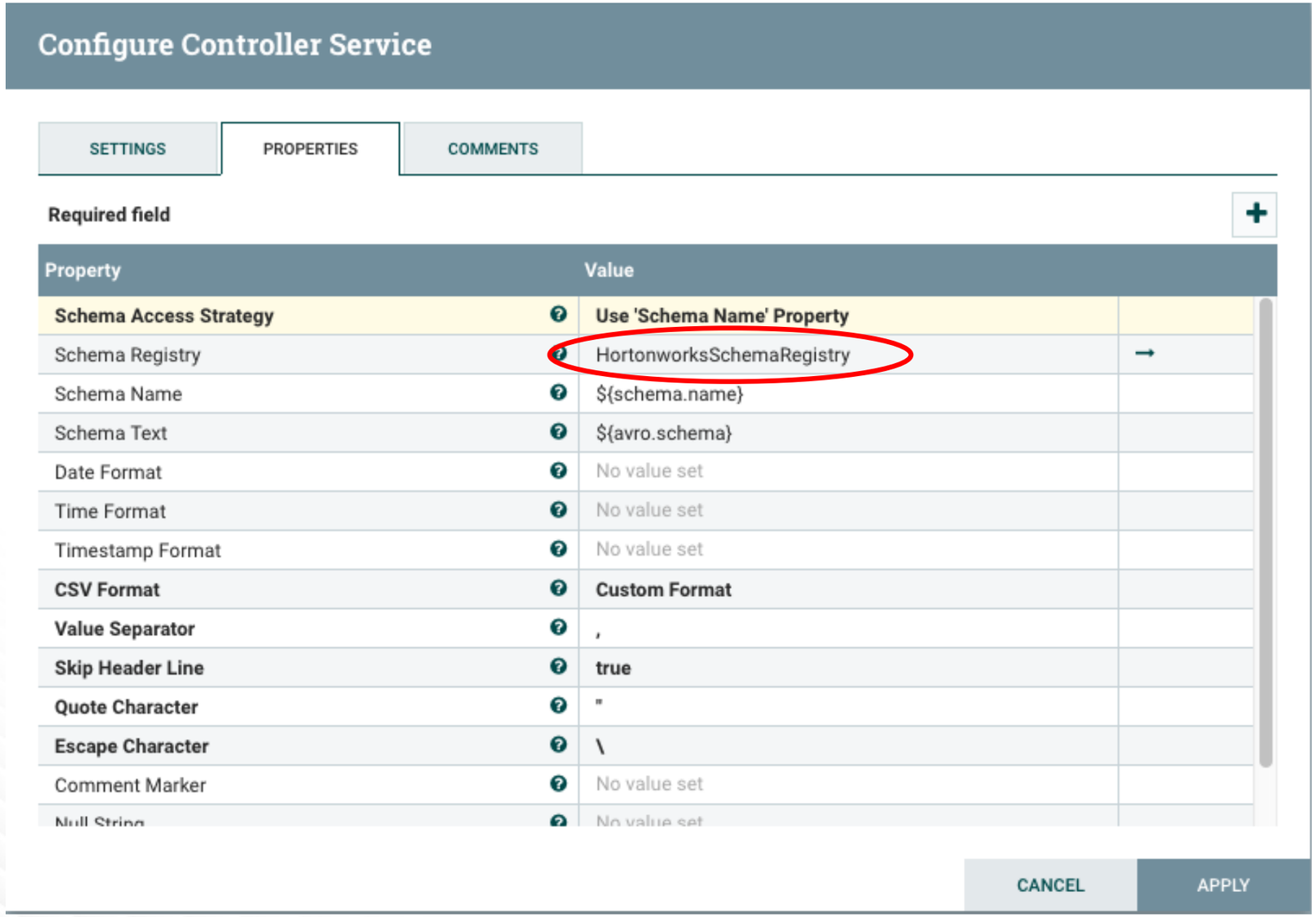




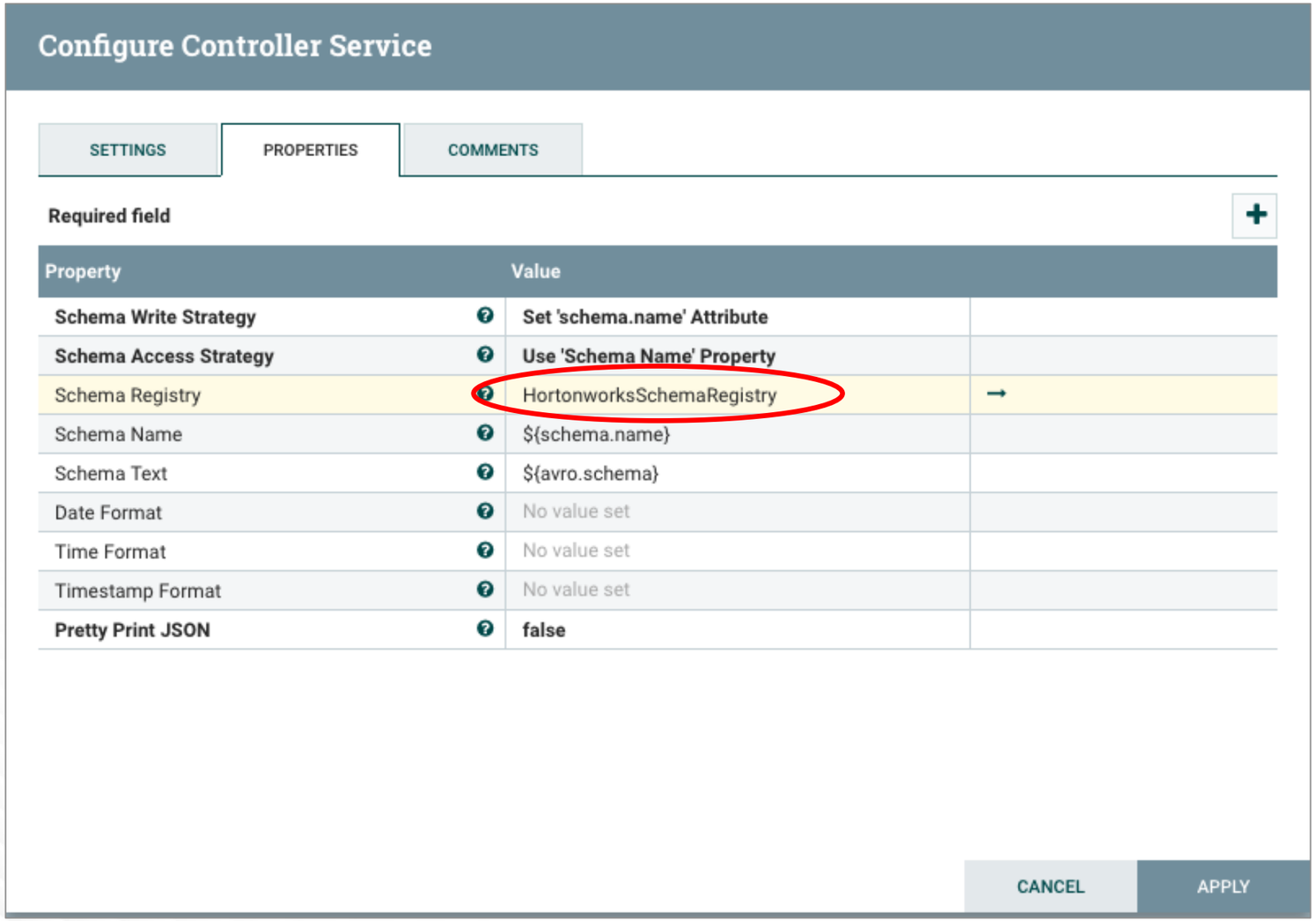
Step 3 – Create HortonworksSchemaRegistry Service



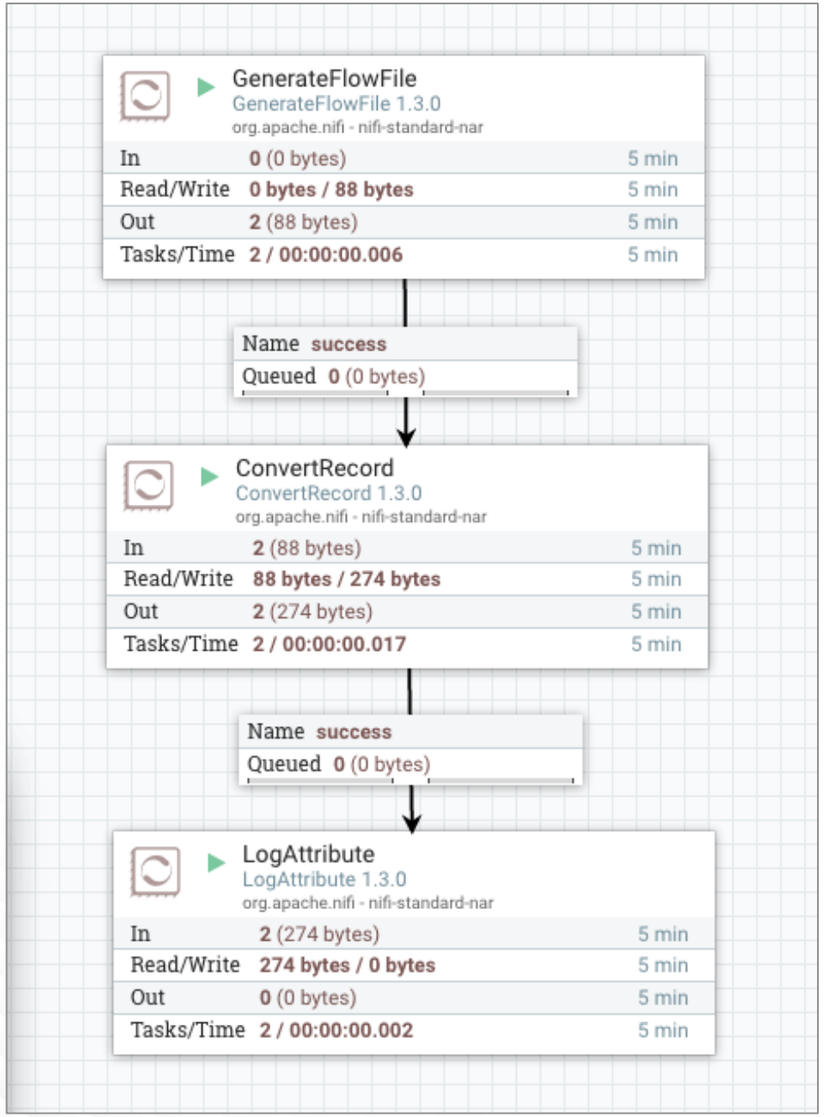
Step 4 – Reconfigure CsvReader



Step 5 – Reconfigure JsonRecordSetWriter



Step 6 – Run the same flow with same results



----------Standard FlowFile Attributes
Key: 'entryDate ' Value: 'Thu Aug 31 13:28:02 EDT 2017’
Key: 'lineageStartDate' Value: 'Thu Aug 31 13:28:02 EDT 2017’
Key: 'fileSize' Value: '137’
FlowFile Attribute Map Content
Key: 'filename' Value: '326844487150210’
Key: 'mime.type' Value: 'application/json’
Key: 'path' Value: './’
Key: 'record.count' Value: ’2’
Key: 'schema.name' Value: 'person’
Key: 'uuid' Value: 'e9198166-0cff-400b-a39d-9c8c9c565f85’

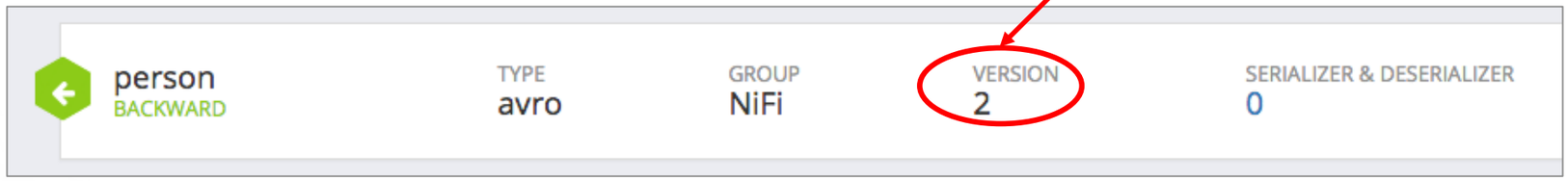
[
{"first_name":"John","last_name":"Smith"},
{"first_name":"Mike","last_name":"Jones"}
]

Example – Use Specific Schema from HWX Schema Registry

Specifying a Schema Version

- Previous example used “Schema Name” for “Schema Access Strategy”
  - NiFi retrieved latest version of schema for name
  - Cached schema based on configuration in controller service
- We can also use “HWX Schema Reference Attributes” to be more specific
  - *schema.identifier*
  - *schema.version*
  - *schema.protocol.version*

Add New Version of Schema

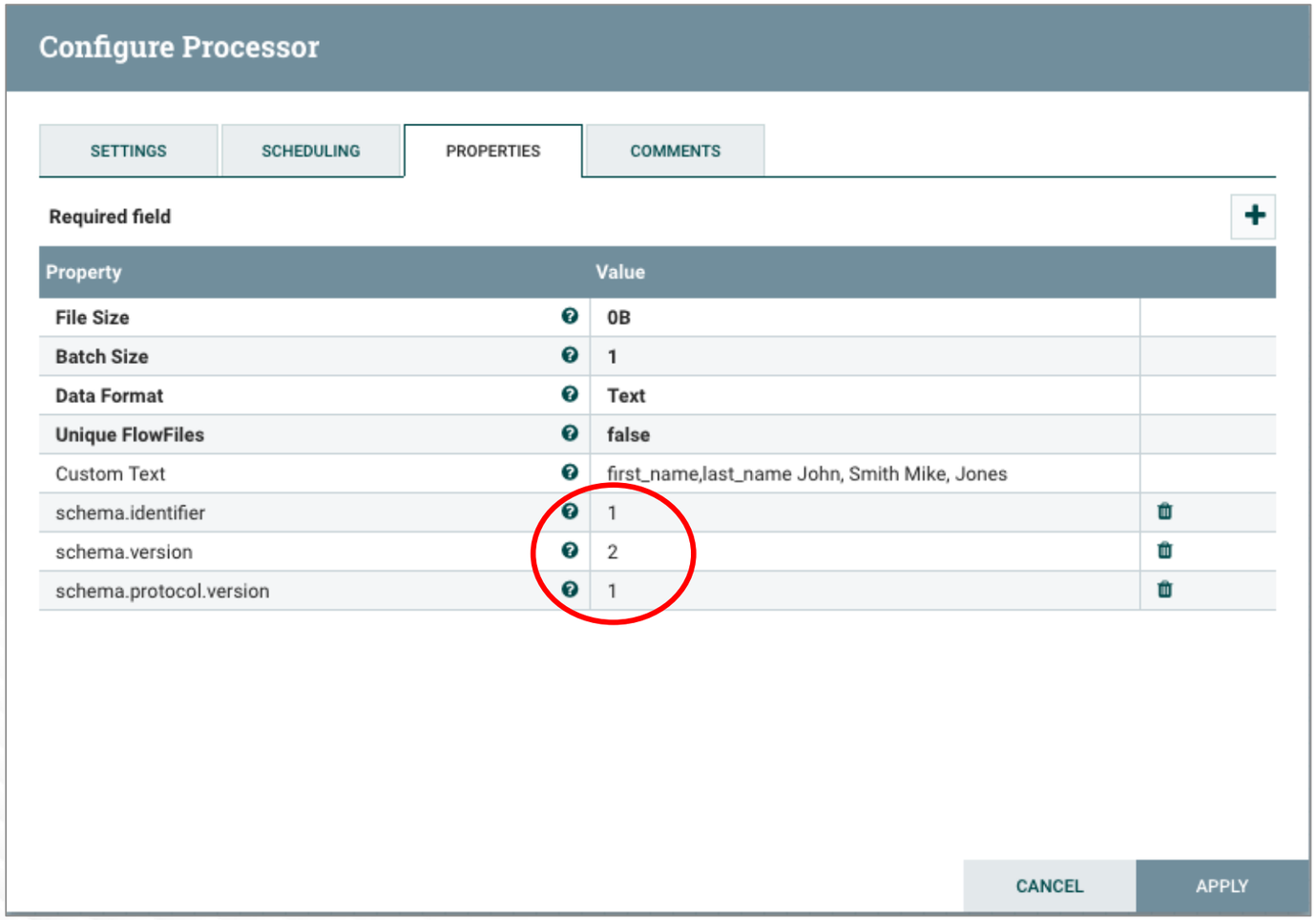


Obtaining Identifier, Version, Protocol

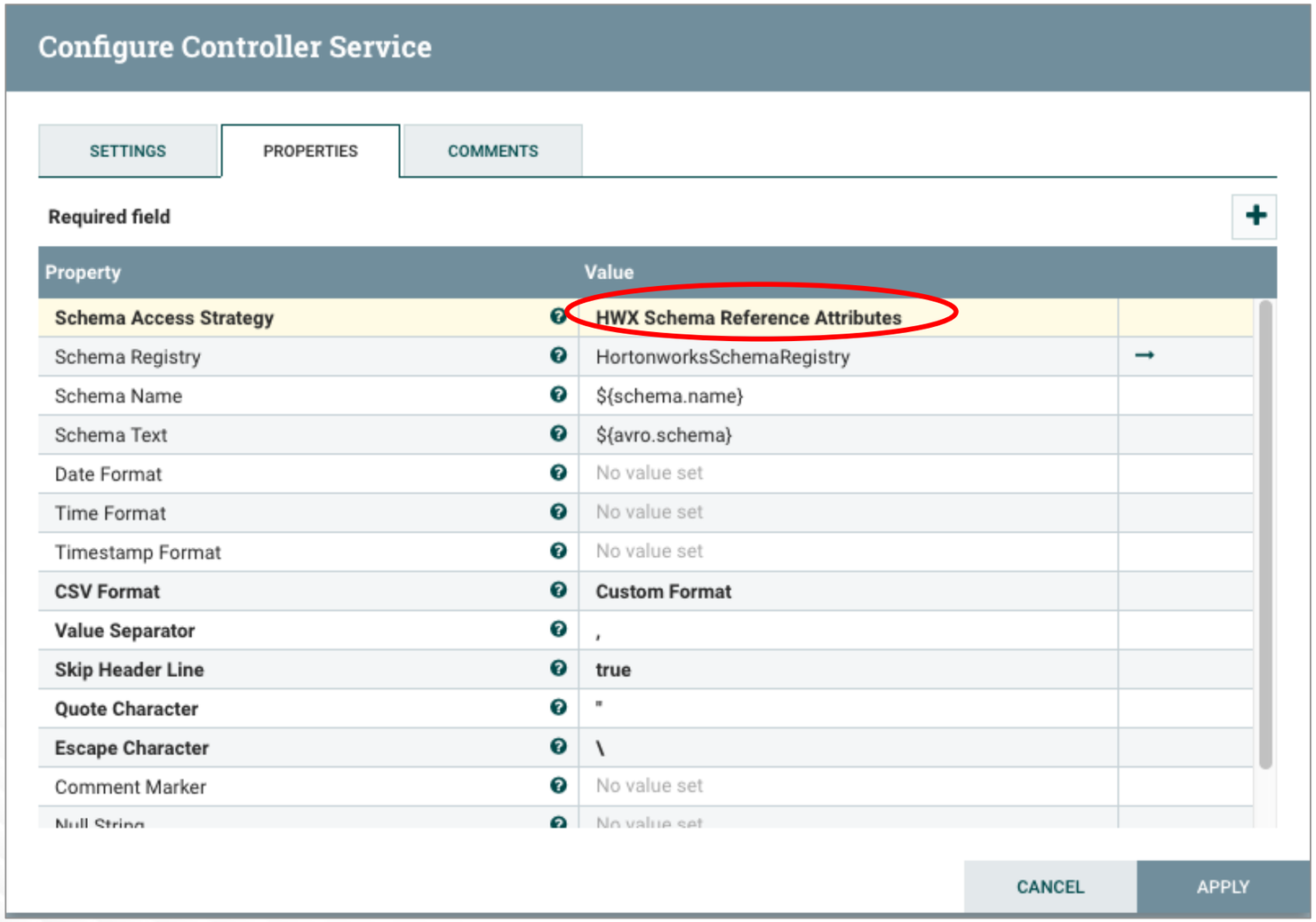
- We can get these values from the schema registry REST API
  - http://localhost:9090/api/v1/schemaregistry/schemas/person
  - http://localhost:9090/api/v1/schemaregistry/schemas/person/versions
  - Protocol Version is always ‘1’ for now

Update Flow to Specify Attributes

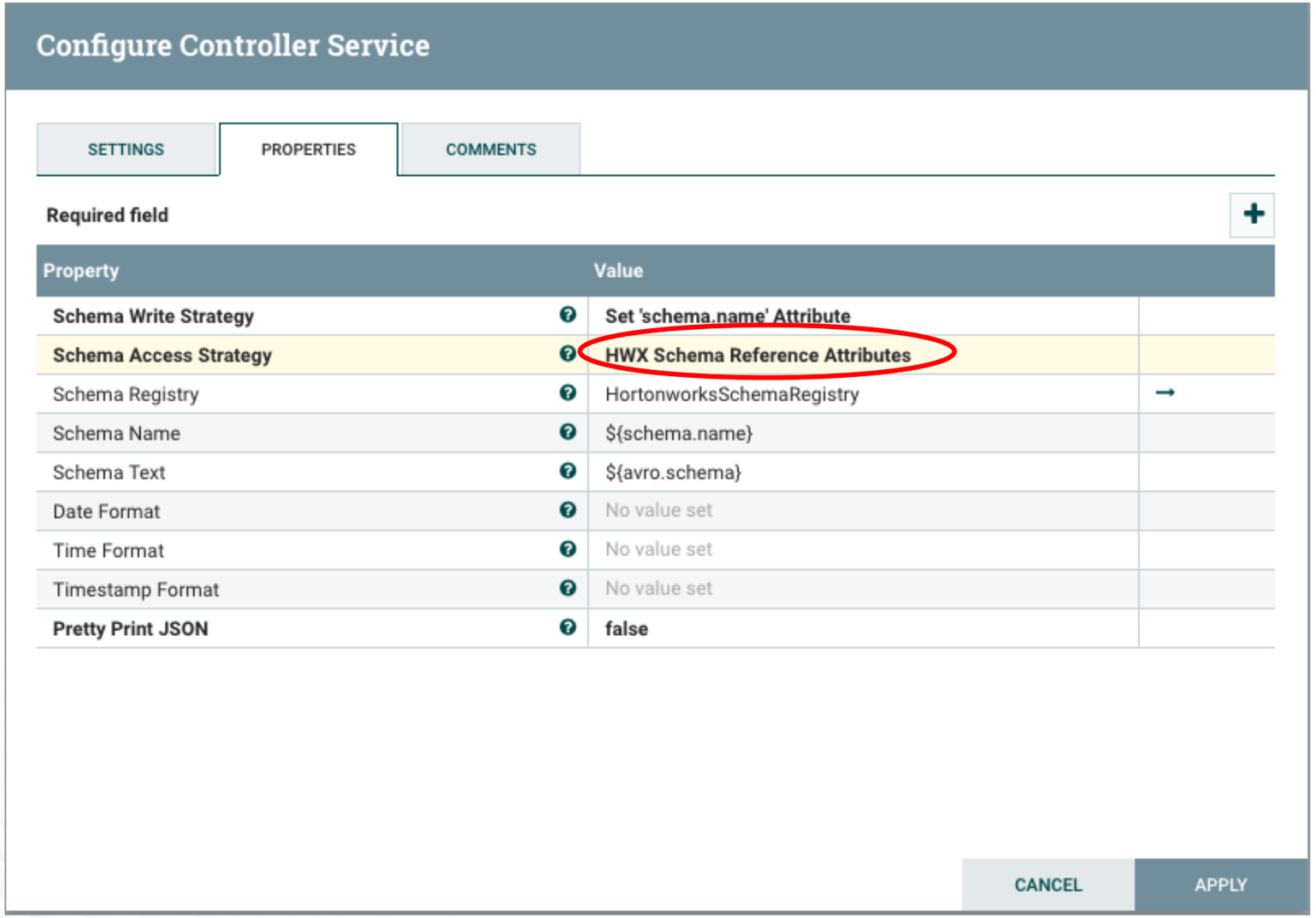
- Remove schema.name and add additional attributes in GenerateFlowFile



Update CsvReader with new Schema Access Strategy



Update JsonRecordSetWriter with new Schema Access Strategy



Run the Flow Again

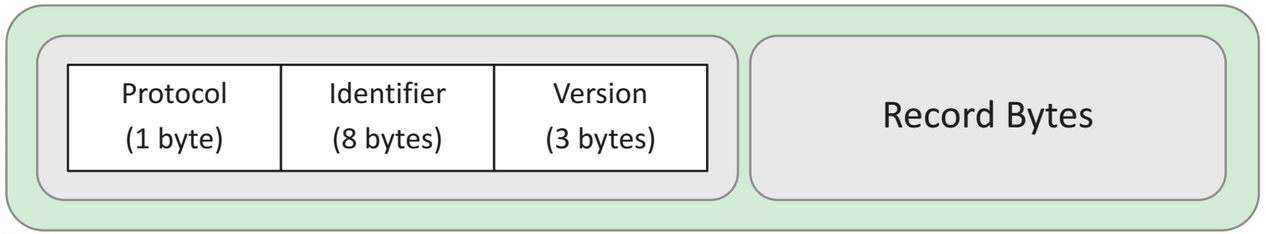
- Using v2 of the schema we should only see first_name:

```
Key: 'schema.identifier'  Value: '1’
Key: 'schema.name'Value: 'person’
Key: 'schema.protocol.version' Value: '1’
Key: 'schema.version'  Value: '2’
Key: 'uuid'  Value: '34407f4e-3bf1-46d5-a6d4-6da5ba197eb8’
-----------------------------------------------------
[{"first_name":"John"},{"first_name":"Mike"}]
```

Apache NiFi + Apache Kafka + HWX Schema Registry

Publishing

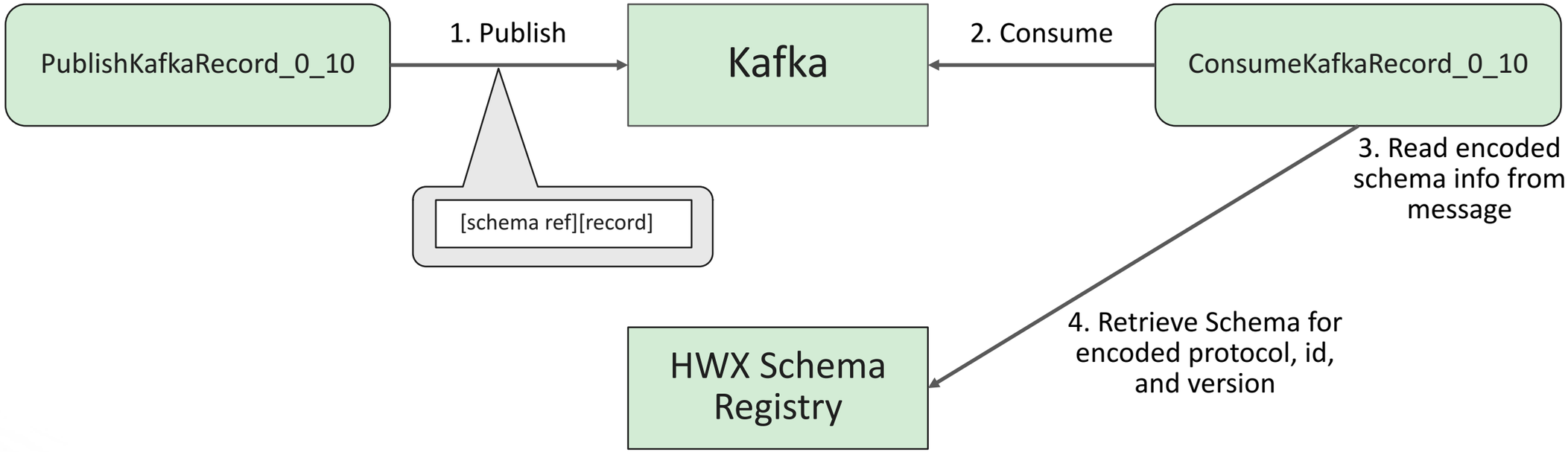
- PublishKafkaRecord_0_10
  - Streams incoming flow file as records using configured RecordReader
  - Serializes each record to bytes using configured RecordSetWriter
- Generally don't want to publish schema on every message
  - "*Schema Write Strategy*" of RecordSetWriter controls where schema ends up
  - "*HWX Content-Encoded Schema Reference*" encodes schema info at beginning of content
  - Single record published as encoded schema reference + bytes of a record



Consuming

- ConsumeKafkaRecord_0_10
  - Reads messages from Kafka into records using configured RecordReader
  - Writes records to a flow file using configured RecordSetWriter
- If publisher used "*HWX Content-Encoded Schema Reference*" as the **Schema Writer Strategy** then consumer needs to use ""*HWX Content-Encoded Schema Reference*" as the **Schema Access Strategy**

Publish & Consume



Additional Resources

- https://blogs.apache.org/nifi/entry/record-oriented-data-with-nifi
- https://blogs.apache.org/nifi/entry/real-time-sql-on-event
- https://community.hortonworks.com/content/kbentry/119766/installing-a-local-hortonworks-registry-to-use-wit.html
- https://community.hortonworks.com/articles/131320/using-partitionrecord-grokreaderjsonwriter-to-pars.html
- https://community.hortonworks.com/articles/115311/convert-csv-to-json-avro-xml-using-convertrecord-p.html