Schema and Format New Design

- Status: Design
- Author: Jiyong Huang
- Discussion: https://github.com/lf-edge/ekuiper/issues/1467

eKuiper Create Stream

- schema schema validation/ protobuf schema protobuf
- Schema schema
- Schemaless schema Flow editor.

PIP Schema Schema Schema API UI

Schema Schema Schema ProtobufAvro eKuiper source Schema

3 Schema

1. Schemaless schema **test schema API Schema schema Flow Editor schema**
2. Schema hint eKuiper source schema JSON source **schema SQL validationFlow Editor**
3. Dynamic schema schemaId Schema Protobuf pb **merge schema Flow Editor**
4. Static schemaSchema **schema descriptor */API***

Source

3 Schema

```
create stream protoDemo (
  id int,
  name string
  ) WITH (FORMAT="protobuf", DATASOURCE="protoDemo",SCHEMAID="helloworld.HelloReply"
```

1. schemaless
2. FORMAT: serilization format JSONBINARYPROTOBUF schema protobuf schema
3. SCHEMAID: 1 schema protobuf schema

Stream schema

schema SQL Flow editor

schema schemaIdstream schema

Format

1. custom (delemitedavro
2. go so schema registry

Serilization Format	Schema Keys	Parse Schema Descriptor
json	optional	json schema (TBD)
binary	n/a	n/a
protobuf	required	proto
custom	required, provided by the extended code	optional, provided by the extended code
delimited (TBD)	n/a, must have stream schema def or the first line should have def	n/a
avro (TBD)	required	avro

Schema Registry

schema soFile soFile so proto

```
###
POST http://{{host}}/schemas/protobuf
Content-Type: application/json

{
  "name": "schema2",
  "file": "file:///C:/repos/go/src/github.com/lfedge/ekuiper/internal/schema/test/test2.proto",
  "soFile": "file:///C:/repos/go/src/github.com/lfedge/ekuiper/internal/schema/test/test2.so"
}
```

Use static protobuf

helloworld.proto

```
syntax = "proto3";

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

1. Generate go code for the pb file, check <https://developers.google.com/protocol-buffers/docs/reference/go-generated> for detail.

   ```
   protoc --go_opt=Mhelloworld.proto=com.main --go_out=. helloworld.proto
   ```

2. Move the generated file `helloworld.pb.go` into the go project `test`. Rename the package to `main`.
3. Create the wrapper file. For each message in the proto, implement 3 functions: Encode, Decode, GetXXX. Example:

   ```
   package main

   func (x *HelloRequest) Encode(d interface{}) ([]byte, error) {
     switch r := d.(type) {
     case map[string]interface{}:
         t, ok := r["name"]
         if ok {
             if v, ok := t.(string); ok {
                 x.Name = v
             } else {
                 return nil, fmt.Errorf("name is not string")
             }
         } else {
             // if required, return error
             fmt.Println("message is not found")
         }
         return proto.Marshal(x)
     default:
         return nil, fmt.Errorf("unsupported type %v, must be a map", d)
     }
   }

   func (x *HelloRequest) Decode(b []byte) (interface{}, error) {
     err := proto.Unmarshal(b, x)
     if err != nil {
     return nil, err
     }
     result := make(map[string]interface{}, 1)
     result["name"] = x.Name
     return result, nil
   }

   func GetHelloRequest() interface{} {
     return &HelloRequest{}
   }
   ```

4. Build the project into a plugin so file. `go build --buildmode=plugin -o helloworld.so .`.
5. Create schema with the .so file.

```
###
POST http://{{host}}/schemas/protobuf
Content-Type: application/json

{
  "name": "helloworld",
  "file": "file:///tmp/helloworld.proto",
  "soFile": "file:///tmp/helloworld.so"
}
```

6. Create the stream to use the static schema.

```
create stream protoDemo () WITH (FORMAT="protobuf", DATASOURCE="protoDemo",SCHEMAID="helloworld.HelloReply"
```

Use custom format

schema custom

1. EncodeDecodeGetXXX
2.
3. so
4. schema registry

```
###
POST http://{{host}}/schemas/custom
Content-Type: application/json

{
  "name": "custom1",
  "soFile": "file:///tmp/custom1.so"
}
```

5. source

```
create stream customDemo () WITH (FORMAT="custom", DATASOURCE="protoDemo",SCHEMAID="custom1.Message"
```

Sink static schema

source sink static schema source sink format schemaId

```
###
POST http://{{host}}/rules
Content-Type: application/json

{
  "id": "rule1",
  "sql": "SELECT * FROM demo",
  "actions": [{
    "mqtt": {
      "server": "tcp://syno.home:1883",
      "topic": "result/protobuf",
      "format": "custom",
      "schemaId": "custom1.Message",
      "sendSingle": true
    }
  }]
}
```