Release H: AVRO

- Introduction
- Apicurio
- AVRO Schema
- AVRO Code Generation
- JAVA Producer/Consumer
- Links

Introduction

Apache Avro is a data serialization system that enables storing JSON files with a schema definition to be stored in binary format.

This increases the efficiency of message throughput/storage in Kafka.

In order to use it we need to use a schema registry like Apicurio.

Apicurio

Apicurio Registry is a schema registry and an API registry, which stores and retrieves event schemas and API designs.

To run it with Strimzi over SSL please we use the following service definition:

Apicurio

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: kafka-registry
  namespace: kafka
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
  authorization:
    type: simple
    acls:
      # Group Id to consume information for the different topics used by the Service Registry.
      # Name equals to metadata.name property in ApicurioRegistry object
      - resource:
          type: group
          name: service-registry
        operation: Read
      # Rules for the Global global-id-topic
      - resource:
          type: topic
          name: global-id-topic
        operation: Read
      - resource:
          type: topic
          name: global-id-topic
        operation: Describe
      - resource:
          type: topic
          name: global-id-topic
        operation: Write
      - resource:
          type: topic
          name: global-id-topic
        operation: Create
      # Rules for the Global storage-topic
      - resource:
          type: topic
          name: storage-topic
        operation: Read
      - resource:
```

```
          type: topic
          name: storage-topic
        operation: Describe
      - resource:
          type: topic
          name: storage-topic
        operation: Write
      - resource:
          type: topic
          name: storage-topic
        operation: Create
      # Rules for the local topics created by our Service Registry instance
      # Prefix value equals to metadata.name property in ApicurioRegistry object
      - resource:
          type: topic
          name: service-registry-
          patternType: prefix
        operation: Read
      - resource:
          type: topic
          name: service-registry-
          patternType: prefix
        operation: Describe
      - resource:
          type: topic
          name: service-registry-
          patternType: prefix
        operation: Write
      - resource:
          type: topic
          name: service-registry-
          patternType: prefix
        operation: Create
      # Rules for the local transactionalsIds created by our Service Registry instance
      # Prefix equals to metadata.name property in ApicurioRegistry object
      - resource:
          type: transactionalId
          name: service-registry-
          patternType: prefix
        operation: Describe
      - resource:
          type: transactionalId
          name: service-registry-
          patternType: prefix
        operation: Write
      # Rules for internal Apache Kafka topics
      - resource:
          type: topic
          name: __consumer_offsets
        operation: Read
      - resource:
          type: topic
          name: __transaction_state
        operation: Read
      # Rules for Cluster objects
      - resource:
          type: cluster
        operation: IdempotentWrite
---
apiVersion: v1
kind: Service
metadata:
  name: kafka-registry
  namespace: kafka
  labels:
    run: kafka-registry
spec:
  type: LoadBalancer
  selector:
    run: kafka-registry
  ports:
```

```
  - port: 8080
    targetPort: 8080
    protocol: TCP
    name: http
    nodePort: 31808
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kafka-registry
  namespace: kafka
  labels:
    run: kafka-registry
spec:
  selector:
    matchLabels:
      run: kafka-registry
  template:
    metadata:
      labels:
        run: kafka-registry
    spec:
      containers:
      - name: kafka-registry
        image: apicurio/apicurio-registry-mem:2.4.2.Final
        imagePullPolicy: IfNotPresent
        env:
        - name : QUARKUS_PROFILE
          value: prod
        - name : KAFKA_BOOTSTRAP_SERVERS
          value: my-cluster-kafka-bootstrap:9093
        - name : APPLICATION_SERVER_HOST
          value: kafka-registry
        - name : APPLICATION_SERVER_PORT
          value: "8080"
        - name : APPLICATION_ID
          value: strimzi-apicurioregistry
        - name : REGISTRY_PROPERTIES_PREFIX
          value: REGISTRY
        - name : REGISTRY_STREAMS_TOPOLOGY_SECURITY_PROTOCOL
          value: SSL
        - name:  REGISTRY_STREAMS_TOPOLOGY_SSL_KEYSTORE_TYPE
          value: PKCS12
        - name : REGISTRY_STREAMS_TOPOLOGY_SSL_KEYSTORE_LOCATION
          value: /etc/ssl/keystore/keystore.p12
        - name: REGISTRY_STREAMS_TOPOLOGY_SSL_KEYSTORE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: kafka-registry
              key: user.password
        - name:  REGISTRY_STREAMS_TOPOLOGY_SSL_TRUSTSTORE_TYPE
          value: PKCS12
        - name : REGISTRY_STREAMS_TOPOLOGY_SSL_TRUSTSTORE_LOCATION
          value: /etc/ssl/truststore/truststore.p12
        - name: REGISTRY_STREAMS_TOPOLOGY_SSL_TRUSTSTORE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: my-cluster-cluster-ca-cert
              key: ca.password
        ports:
        - name: http
          containerPort: 8080
        volumeMounts:
        - name: keystore
          mountPath: "/etc/ssl/keystore"
          readOnly: true
        - name: truststore
          mountPath: "/etc/ssl/truststore"
          readOnly: true
      volumes:
      - name: keystore
```

```
        secret:
          secretName: kafka-registry
          items:
          - key: user.p12
            path: keystore.p12
      - name: truststore
        secret:
          secretName: my-cluster-cluster-ca-cert
          items:
          - key: ca.p12
            path: truststore.p12
      restartPolicy: Always
```

AVRO Schema

There are various tools available to help us define our AVRO schema (see link below).

Here is the schema definition for the PM data file:

PM Schema

```
{
   "type":"record",
   "name":"PMData",
   "namespace":"org.oran.avro.demo.pm",
    "fields":[
      {
         "name":"event",
         "type":{
            "type":"record",
            "name":"Event",
            "fields":[
               {
                  "name":"commonEventHeader",
                  "type":{
                     "type":"record",
                     "name":"CommonEventHeader",
                     "fields":[
                        {
                           "name":"domain",
                           "type":"string"
                        },
                        {
                           "name": "eventId",
                           "type":"string"
                        },
                        {
                           "name":"eventName",
                           "type":"string"
                        },
                        {
                           "name":"sequence",
                           "type": "int"
                        },
                        {
                           "name":"reportingEntityName",
                           "type":"string"
                        },
                        {
                           "name":"vesEventListenerVersion",
                           "type":"string"
                        },
                        {
                           "name":"version",
                           "type":"string"
                        },
```

```
            {
               "name":"sourceName",
               "type":"string"
            },
            {
               "name":"priority",
               "type":"string"
            },
            {
               "name":"lastEpochMicrosec",
               "type":"long"
            },
            {
               "name":"startEpochMicrosec",
               "type":"long"
            },
            {
               "name":"timeZoneOffset",
               "type":"string"
            }
         ]
      }
   },
   {
      "name":"perf3gppFields",
      "type":{
         "type":"record",
         "name":"Perf3gppFields",
         "fields":[
            {
               "name":"perf3gppFieldsVersion",
               "type":"string"
            },
            {
               "name":"measDataCollection",
               "type":{
                  "type":"record",
                  "name":"MeasDataCollection",
                  "fields":[
                     {
                        "name":"granularityPeriod",
                        "type":"int"
                     },
                     {
                        "name":"measuredEntityDn",
                        "type":"string"
                     },
                     {
                        "name":"measuredEntitySoftwareVersion",
                        "type":"string"
                     },
                     {
                        "name":"measuredEntityUserName",
                        "type":"string"
                     },
                     {
                        "name":"measInfoList",
                        "type":{
                           "type":"array",
                           "items":{
                              "type":"record",
                              "name":"MeasInfoList",
                              "fields":[
                                 {
                                    "name":"measInfoId",
                                    "type":{
                                       "type":"record",
                                       "name":"MeasInfoId",
                                       "fields":[
                                          {
                                             "name":"sMeasInfoId",
```

```
                                                "type":"string"
                                             }
                                          ]
                                       }
                                    },
                                    {
                                       "name":"measTypes",
                                       "type":{
                                          "type":"record",
                                          "name":"MeasTypes",
                                          "fields":[
                                             {
                                                "name":"sMeasTypesList",
                                                "type":{
                                                   "type":"array",
                                                   "items":"string"
                                                }
                                             }
                                          ]
                                       }
                                    },
                                    {
                                       "name":"measValuesList",
                                       "type":{
                                          "type":"array",
                                          "items":{
                                             "type":"record",
                                             "name":"MeasValuesList",
                                             "fields":[
                                                {
                                                   "name":"measObjInstId",
                                                   "type":"string"
                                                },
                                                {
                                                   "name":"suspectFlag",
                                                   "type":"string"

},

                                                {
                                                   "name":"measResults",
                                                   "type":{
                                                      "type":"array",
                                                      "items":{
                                                         "type":"record",
                                                         "name":"MeasResult",
                                                         "fields":[
                                                            {
                                                               "name":"p",
                                                               "type":"int"
                                                            },
                                                            {
                                                               "name":"sValue",
                                                               "type":"string"
                                                            }
                                                         ]
                                                      }
                                                   }
                                                }
                                             ]
                                          }
                                       }
                                    }
                                 ]
                              }
                           }
                        ]
                     }
                  }
               ]
            }
```

```
                }
            ]
        }
    }
  ]
}
```

Note: Every field in the AVRO schema is mandatory. Please remove any fields not included in the JSON file.

AVRO Code Generation

You can generate java classes from your schema using th avro-tools jar.

e.g.

Code Generation

```
java -jar /mnt/c/Users/ktimoney/Downloads/avro-tools-1.11.1.jar  compile schema ./src/main/resources/schemas/pm-all.avsc ./src/main/java/
```

This will produce the following classes in the org/oran/avro/demo/pm/ directory (same as namespace in the schema)

CommonEventHeader.java
Event.java
MeasDataCollection.java
MeasInfoId.java
MeasInfoList.java
MeasResult.java
MeasTypes.java
MeasValuesList.java
Perf3gppFields.java
PMData.java

Note: We need to add some extra annotation to some of the fields in the generated classes

```
@JsonProperty("sMeasInfoId")
private java.lang.CharSequence sMeasInfoId;

@JsonProperty("sMeasTypesList")
private java.util.List<java.lang.CharSequence> sMeasTypesList;

@JsonProperty("sValue")
private java.lang.CharSequence sValue;
```

It has problems using fields starting with a lowercase character followed by an uppercase character.

JAVA Producer/Consumer

AVRO Producer/Consumer

```
package org.oran.avro.demo;

import com.fasterxml.jackson.databind.ObjectMapper;
import io.apicurio.registry.rest.v2.beans.IfExists;
import io.apicurio.registry.serde.SerdeConfig;
import io.apicurio.registry.serde.avro.AvroKafkaDeserializer;
import io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient;
import io.confluent.kafka.schemaregistry.client.rest.RestService;
import io.confluent.kafka.serializers.KafkaAvroSerializer;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Collections;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericDatumReader;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.DatumReader;
import org.apache.avro.io.Decoder;
import org.apache.avro.io.DecoderFactory;
import org.apache.commons.io.IOUtils;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.config.SslConfigs;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.oran.avro.demo.pm.MeasInfoList;
import org.oran.avro.demo.pm.MeasResult;
import org.oran.avro.demo.pm.MeasTypes;
import org.oran.avro.demo.pm.MeasValuesList;
import org.oran.avro.demo.pm.PMData;
import org.oran.avro.demo.pm.PMFlat;


public class PMDataAvroExample {

    private static final String REGISTRY_URL = "http://localhost:8080/apis/registry/v2";
    private static final String CCOMPAT_API_URL = "http://localhost:8080/apis/ccompat/v6";
    private static final String BOOTSTRAP_HOST = "192.168.49.2";
    private static final String BOOTSTRAP_PORT = "30053";
    private static final String SERVERS = BOOTSTRAP_HOST+":"+BOOTSTRAP_PORT;
    private static final String TOPIC_NAME = PMDataAvroExample.class.getSimpleName();
    private static final String SUBJECT_NAME = "key";
    private static final String PASSWORD = "GkELEyyxccrp";
    private static final String KEYSTORE = "/mnt/c/Users/ktimoney/go/kafka/users/user-keystore.jks";
    private static final String TRUSTSTORE = "/mnt/c/Users/ktimoney/go/kafka/users/user-truststore.jks";
    private static final int NUMBER_OF_MESSAGES = 10;


    public static final void main(String [] args) throws Exception {
      PMDataAvroExample app = new PMDataAvroExample();
      //String jsonFile =  "pm_report.json";
      String jsonFile =  "A20000626.2315+0200-2330+0200_HTTPS-6-73.json";
      String jsonData = app.getFileAsString(jsonFile);


        System.out.println("Starting example " + PMDataAvroExample.class.getSimpleName());
        String topicName = TOPIC_NAME;
        String subjectName = SUBJECT_NAME;
        String artifactId = topicName + "-PMData";

        String schemaData = app.getFileAsString("schemas/pm-all.avsc");

        // Create the producer.
        Producer<String, Object> producer = createKafkaProducer();
        int producedMessages = 0;
        // Produce messages
        try {
            System.out.println("Producing ("+NUMBER_OF_MESSAGES+") messages.");
            for (int idx = 0; idx < NUMBER_OF_MESSAGES; idx++) {
                // Use the schema to create a record
                GenericRecord record = parseJson(jsonData, schemaData);

                // Send/produce the message on the Kafka Producer
```

```
                ProducerRecord<String, Object> producedRecord = new ProducerRecord<>(topicName, subjectName,
record);
                producer.send(producedRecord);

                Thread.sleep(100);
            }
            System.out.println(NUMBER_OF_MESSAGES + " messages sent successfully.");
        } finally {
            System.out.println("Closing the producer.");
            producer.flush();
            producer.close();
        }

        // Create the consumer
        System.out.println("Creating the consumer.");
        KafkaConsumer<Long, GenericRecord> consumer = createKafkaConsumer();

        // Subscribe to the topic
        System.out.println("Subscribing to topic " + topicName);
        consumer.subscribe(Collections.singletonList(topicName));

        // Consume messages.
        try {
            int messageCount = 0;
            System.out.println("Consuming ("+NUMBER_OF_MESSAGES+") messages.");
            while (messageCount < NUMBER_OF_MESSAGES) {
                final ConsumerRecords<Long, GenericRecord> records = consumer.poll(Duration.ofSeconds(1));
                messageCount += records.count();
                if (records.count() == 0) {
                    // Wait for messages to become available.
                    System.out.println("Waiting for messages...");
                } else records.forEach(record -> {
                    GenericRecord recordValue = record.value();
                    System.out.println("Consumed a message: ");
                    ObjectMapper mapper = new ObjectMapper();
                    try {
                      PMData pms =  mapper.readValue(String.valueOf(record.value()), PMData.class);
                      List<PMFlat> flatList = transformPMS(pms);
                      flatList.forEach(System.out::println);
                    } catch (Exception e) {
                      e.printStackTrace();
                    }

                });
            }
        } finally {
            System.out.println("Closing the consumer.");
            consumer.close();
        }

        System.out.println("Done (success).");
    }

    /**
     * Creates the Kafka producer.
     */
    private static Producer<String, Object> createKafkaProducer() {
        Properties props = new Properties();

        // Configure kafka settings
        props.putIfAbsent(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, SERVERS);
        props.putIfAbsent(ProducerConfig.CLIENT_ID_CONFIG, "Producer-" + TOPIC_NAME);
        props.putIfAbsent(ProducerConfig.ACKS_CONFIG, "all");

        // Configure Service Registry location
        props.putIfAbsent(SerdeConfig.REGISTRY_URL, REGISTRY_URL);
        // Get an existing schema - do not auto-register the schema if not found.
        props.putIfAbsent(SerdeConfig.AUTO_REGISTER_ARTIFACT, Boolean.TRUE);
        props.putIfAbsent(SerdeConfig.AUTO_REGISTER_ARTIFACT_IF_EXISTS, IfExists.RETURN.name());

        //Just if security values are present, then we configure them.
```

```
        configureSecurityIfPresent(props);

        RestService restService = new RestService(CCOMPAT_API_URL);
        final Map<String, String> restServiceProperties = new HashMap<>();
        CachedSchemaRegistryClient schemaRegistryClient = new CachedSchemaRegistryClient(restService, 100,
restServiceProperties);

        Map<String, String> properties = new HashMap<>();

        // Configure Service Registry location (Confluent API)
        properties.put("schema.registry.url", CCOMPAT_API_URL);
        properties.put("auto.register.schemas", "true");
        // Map the topic name to the artifactId in the registry
        properties.put("value.subject.name.strategy", "io.confluent.kafka.serializers.subject.
TopicRecordNameStrategy");

        // Use the Confluent provided Kafka Serializer for Avro
        KafkaAvroSerializer valueSerializer = new KafkaAvroSerializer(schemaRegistryClient, properties);
        StringSerializer keySerializer = new StringSerializer();

        // Create the Kafka producer
        Producer<String, Object> producer = new KafkaProducer<String, Object>(props, keySerializer,
valueSerializer);

        return producer;
    }

    /**
     * Creates the Kafka consumer.
     */
    private static KafkaConsumer<Long, GenericRecord> createKafkaConsumer() {
        Properties props = new Properties();

        // Configure Kafka
        props.putIfAbsent(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, SERVERS);
        props.putIfAbsent(ConsumerConfig.GROUP_ID_CONFIG, "Consumer-" + TOPIC_NAME);
        props.putIfAbsent(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        props.putIfAbsent(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        props.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        props.putIfAbsent(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        // Use the Apicurio Registry provided Kafka Deserializer for Avro
        props.putIfAbsent(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, AvroKafkaDeserializer.class.
getName());

        // Configure Service Registry location
        props.putIfAbsent(SerdeConfig.REGISTRY_URL, REGISTRY_URL);
        // Enable "Confluent Compatible API" mode in the Apicurio Registry deserializer
        props.putIfAbsent(SerdeConfig.ENABLE_CONFLUENT_ID_HANDLER, Boolean.TRUE);

        //Just if security values are present, then we configure them.
        configureSecurityIfPresent(props);

        // Create the Kafka Consumer
        KafkaConsumer<Long, GenericRecord> consumer = new KafkaConsumer<>(props);
        return consumer;
    }


    private static void configureSecurityIfPresent(Properties props) {
      props.putIfAbsent(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SSL");
      props.putIfAbsent(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, TRUSTSTORE);
      props.putIfAbsent(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG,  PASSWORD);

      // configure the following three settings for SSL Authentication
      props.putIfAbsent(SslConfigs.SSL_KEYSTORE_LOCATION_CONFIG, KEYSTORE);
      props.putIfAbsent(SslConfigs.SSL_KEYSTORE_PASSWORD_CONFIG, PASSWORD);
      props.putIfAbsent(SslConfigs.SSL_KEY_PASSWORD_CONFIG, PASSWORD);
    }


    private static GenericData.Record parseJson(String json, String schema) throws IOException {
```

```
      Schema parsedSchema = new Schema.Parser().parse(schema);
      Decoder decoder = DecoderFactory.get().jsonDecoder(parsedSchema, json);

      DatumReader<GenericData.Record> reader =
          new GenericDatumReader<>(parsedSchema);
      return reader.read(null, decoder);
    }

    public String getFileAsString(String fileName)throws Exception
    {
      ClassLoader classLoader = getClass().getClassLoader();
      InputStream inputStream = classLoader.getResourceAsStream(fileName);

      // the stream holding the file content
      if (inputStream == null) {
          throw new IllegalArgumentException("file not found! " + fileName);
      } else {
          String result = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
          return result;
      }
    }


    public static List<PMFlat> transformPMS(PMData pms){
      List<PMFlat> flatList = new ArrayList<>();
      // Get the constant values
      String domain = pms.getEvent().getCommonEventHeader().getDomain().toString();
      String eventName = pms.getEvent().getCommonEventHeader().getEventName().toString();
      String sourceName = pms.getEvent().getCommonEventHeader().getSourceName().toString();
      long startEpochMicrosec = pms.getEvent().getCommonEventHeader().getStartEpochMicrosec();
      long lastEpochMicrosec = pms.getEvent().getCommonEventHeader().getLastEpochMicrosec();
      String timeZoneOffset = pms.getEvent().getCommonEventHeader().getTimeZoneOffset().toString();
      int granularityPeriod = pms.getEvent().getPerf3gppFields().getMeasDataCollection().getGranularityPeriod();
      String measuredEntityDn = pms.getEvent().getPerf3gppFields().getMeasDataCollection().
getMeasuredEntityDn().toString();
      String measuredEntityUserName = pms.getEvent().getPerf3gppFields().getMeasDataCollection().
getMeasuredEntityUserName().toString();

      List<MeasInfoList> measInfoLists = pms.getEvent().getPerf3gppFields().getMeasDataCollection().
getMeasInfoList();
      // Loop through the measurements
      for(MeasInfoList measInfoList: measInfoLists) {
        String sMeasInfoId = measInfoList.getMeasInfoId().getSMeasInfoId().toString();
        MeasTypes measTypes = measInfoList.getMeasTypes();
        List<CharSequence> sMeasTypesList = measTypes.getSMeasTypesList();
        List<MeasValuesList> measValuesLists = measInfoList.getMeasValuesList();
        for(MeasValuesList measValuesList: measValuesLists) {
          String measObjInstId = measValuesList.getMeasObjInstId().toString();
          String suspectFlag = measValuesList.getSuspectFlag().toString();
          List<MeasResult> measResultList = measValuesList.getMeasResults();
          for(MeasResult measResult: measResultList) {
            // Create new PMSFlat object
            PMFlat flat = new PMFlat();
            flat.setDomain(domain);
            flat.setEventName(eventName);
            flat.setSourceName(sourceName);
            flat.setStartEpochMicrosec(startEpochMicrosec);
            flat.setLastEpochMicrosec(lastEpochMicrosec);
            flat.setTimeZoneOffset(timeZoneOffset);
            flat.setGranularityPeriod(granularityPeriod);
            flat.setMeasuredEntityDn(measuredEntityDn);
            flat.setMeasuredEntityUserName(measuredEntityUserName);
            flat.setSMeasInfoId(sMeasInfoId);
            flat.setMeasObjInstId(measObjInstId);
            flat.setSuspectFlag(suspectFlag);

            String sMeasType = sMeasTypesList.get(measResult.getP()-1).toString();
            flat.setSMeasType(sMeasType);
            String sValue = measResult.getSValue().toString();
            flat.setSValue(sValue);
            // add the object to the list
```

```
            flatList.add(flat);
          }
        }
      }
      return flatList;
    }

}
```

Note: Extra configuration is required to make the the message(s) compatible with the confluent format (CCOMPAT_API_URL).

We unmarshall the message to the generated code classes and then map to the PMFlat class.

```
package org.oran.avro.demo.pm;
import lombok.Data;
import lombok.ToString;
@Data
@ToString
public class PMFlat {
  private String domain;
  private String eventName;
  private String sourceName;
  private long startEpochMicrosec;
  private long lastEpochMicrosec;
  private String timeZoneOffset;
  private int granularityPeriod;
  private String measuredEntityDn;
  private String measuredEntityUserName;
  private String sMeasInfoId;
  private String measObjInstId;
  private String suspectFlag;
  private String sMeasType;
  private String sValue;
}
```

Links

Apache Avro - a data serialization system

Apicurio Registry

AVRO SCHEMA TOOLS

Example applications using the Apicurio Registry

AVRO Getting Started (Java)