Release H: Protobuf

Introduction

Protobuf is data format used to serialize structured data.

Similar to AVRO it can be used to serialize JSON data and store it in binary format.

This help with transfer speed and data storage.

Protobuf Schema

We can use a protobuf schema with Apicurio registry that same way as we do with AVRO.

PM Data proto

```
syntax = "proto3";

option java_package = "org.oran.protobuf.demo.pm";
option java_outer_classname = "PMProtos";

message PMDataType {
  Event event = 1;
}

message Event {
   CommonEventHeader commonEventHeader = 1;
   Perf3gppFields perf3gppFields = 2;
}

message CommonEventHeader {
    string domain = 1;
    string eventName = 2;
    string sourceName = 3;
    string reportingEntityName = 4;
    int64 startEpochMicrosec = 5;
    int64 lastEpochMicrosec = 6;
    string timeZoneOffset = 7;
    string eventId = 8;
    int32 sequence = 9;
    string priority = 10;
    string version = 11;
    string vesEventListenerVersion = 12;
}

message Perf3gppFields {
    string perf3gppFieldsVersion = 1;
    MeasDataCollection measDataCollection = 2;
}

message MeasDataCollection {
    int32 granularityPeriod = 1;
    string measuredEntityUserName = 2;
    string measuredEntityDn = 3;
    string measuredEntitySoftwareVersion = 4;
    repeated MeasInfoList measInfoList = 5;
}

message MeasInfoList {
    MeasInfoId measInfoId = 1;
    MeasTypes measTypes = 2;
    repeated MeasValuesList measValuesList = 3;
}

message MeasInfoId {
    string sMeasInfoId = 1;
}

message MeasTypes {
    repeated string sMeasTypesList = 1;
}

message MeasValuesList  {
    string measObjInstId = 1;
    string suspectFlag = 2;
    repeated MeasResults measResults = 3;
}

message MeasResults {
    int32 p = 1;
    string sValue = 2;
}
```

Protobuf Maven Plugin

We can use the protoc-jar-maven-plugin plugin to automatically generated the corresponding java class for our .proto file.

protoc-jar-maven-plugin

```
<plugin>
    <groupId>com.github.os72</groupId>
    <artifactId>protoc-jar-maven-plugin</artifactId>
    <version>3.11.4</version>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
                <inputDirectories>
                    <include>${project.basedir}/src/main/proto</include>
                </inputDirectories>
                <includeMavenTypes>direct</includeMavenTypes>
                <outputTargets>
                    <outputTarget>
                        <type>java</type>
                        <addSources>main</addSources>
                        <outputDirectory>
                          ${project.basedir}/target/generated-sources/protobuf
                      </outputDirectory>
                    </outputTarget>
                </outputTargets>
            </configuration>
        </execution>
    </executions>
</plugin>
```

The auto generated java class will be located in "${project.basedir}/target/generated-sources/protobuf" + "org.oran.protobuf.demo.pm" (option java_package = "org.oran.protobuf.demo.pm" from the proto file)

Java

The following class shows how to covert a JSON file to the protobuf format, send it to Kafka, read it back and unmarshall it with the auto generated proto class.

The nested records are then transformed to a flat data structure.

```
package org.oran.protobuf.demo;

import com.google.protobuf.InvalidProtocolBufferException;
import com.google.protobuf.Message;
import com.google.protobuf.MessageOrBuilder;
import com.google.protobuf.Struct;
import com.google.protobuf.Struct.Builder;
import com.google.protobuf.util.JsonFormat;
import org.oran.protobuf.demo.pm.PMProtos.*;
import io.apicurio.registry.serde.SerdeConfig;
import io.apicurio.registry.serde.protobuf.ProtobufKafkaDeserializer;
import io.apicurio.registry.serde.protobuf.ProtobufKafkaSerializer;
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.List;
```

```
import java.util.Properties;
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.PMFlat;


public class PMDataProtobufExample {

    private static final String REGISTRY_URL = "http://localhost:8080/apis/registry/v2";
    private static final String BOOTSTRAP_HOST = "192.168.49.2";
    private static final String BOOTSTRAP_PORT = "31809";
    private static final String SERVERS = BOOTSTRAP_HOST+":"+BOOTSTRAP_PORT;
    private static final String PASSWORD = "Yx79VGQAWIMu";
    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;
    private static final String TOPIC_NAME = PMDataProtobufExample.class.getSimpleName();
    private static final String SCHEMA_NAME = "PMData";


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


        System.out.println("Starting example " + PMDataProtobufExample.class.getSimpleName());
        String topicName = TOPIC_NAME;
        String key = SCHEMA_NAME;

        // Create the producer.
        Producer<Object, PMDataType> producer = createKafkaProducer();
        // Produce 2 messages.
        try {
                System.out.println("Producing (1) messages.");
                // Send/produce the message on the Kafka Producer
                ProducerRecord<Object, PMDataType> producedRecord = new ProducerRecord<>(topicName, key, pm);
                producer.send(producedRecord);
            System.out.println("Messages successfully produced.");
        } finally {
            System.out.println("Closing the producer.");
            producer.flush();
            producer.close();
        }

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

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

        // Consume the message.
        try {
            int messageCount = 0;
            System.out.println("Consuming (1) messages.");
            while (messageCount < 1) {
                final ConsumerRecords<Long, PMDataType> records = consumer.poll(Duration.ofSeconds(1));
```

```
                messageCount += records.count();
                if (records.count() == 0) {
                    // Do nothing - no messages waiting.
                    System.out.println("No messages waiting...");
                } else records.forEach(record -> {
                    PMDataType value = record.value();
                    System.out.println("Consumed a message:  " );
                    // unmarshall the message
                    List<PMFlat> flatList = transformPMS(value);
                    flatList.forEach(System.out::println);
                });
            }
        } finally {
            consumer.close();
        }

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

    /**
     * Creates the Kafka producer.
     */
    private static Producer<Object, PMDataType> 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");
        props.putIfAbsent(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        // Use the Apicurio Registry provided Kafka Serializer for Protobuf
        props.putIfAbsent(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, ProtobufKafkaSerializer.class.
getName());

        // Configure Service Registry location
        props.putIfAbsent(SerdeConfig.REGISTRY_URL, REGISTRY_URL);
        props.putIfAbsent(SerdeConfig.EXPLICIT_ARTIFACT_GROUP_ID, "default");

        // Register the artifact if not found in the registry.
        props.putIfAbsent(SerdeConfig.AUTO_REGISTER_ARTIFACT, Boolean.TRUE);

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

        // Create the Kafka producer
        Producer<Object, PMDataType> producer = new KafkaProducer<>(props);
        return producer;
    }

    /**
     * Creates the Kafka consumer.
     */
    private static KafkaConsumer<Long, PMDataType> 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 Protobuf
        props.putIfAbsent(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, ProtobufKafkaDeserializer.class.
getName());

        // Configure Service Registry location
        props.putIfAbsent(SerdeConfig.REGISTRY_URL, REGISTRY_URL);

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

```
    // Create the Kafka Consumer
    KafkaConsumer<Long, PMDataType> 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);
}

public static Message fromJson(String json) throws IOException {
  Builder structBuilder = Struct.newBuilder();
  JsonFormat.parser().ignoringUnknownFields().merge(json, structBuilder);
  return structBuilder.build();
}

@SuppressWarnings({"unchecked", "rawtypes"})
public static PMDataType fromJsonToClass(String json) {
  PMDataType.Builder pmBuilder = PMDataType.newBuilder();
  try {
       JsonFormat.parser().merge(json, pmBuilder);
  } catch(InvalidProtocolBufferException e) {
    e.printStackTrace();
  }
  PMDataType value = pmBuilder.build();
  return value;
}

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(PMDataType pms){
  List<PMFlat> flatList = new ArrayList<>();
  Event event = pms.getEvent();
  CommonEventHeader commonEventHeader = event.getCommonEventHeader();
  String domain = commonEventHeader.getDomain();
  String eventName = commonEventHeader.getEventName();
  String sourceName = commonEventHeader.getSourceName();
  String reportingEntityName = commonEventHeader.getReportingEntityName();
  long startEpochMicrosec = commonEventHeader.getStartEpochMicrosec();
  long lastEpochMicrosec = commonEventHeader.getLastEpochMicrosec();
  String timeZoneOffset = commonEventHeader.getTimeZoneOffset();
  Perf3gppFields perf3gppFields = event.getPerf3GppFields();
  String perf3gppFieldsVersion = perf3gppFields.getPerf3GppFieldsVersion();
  MeasDataCollection measDataCollection = perf3gppFields.getMeasDataCollection();
  int granularityPeriod = measDataCollection.getGranularityPeriod();
  String measuredEntityUserName = measDataCollection.getMeasuredEntityUserName();
  String measuredEntityDn = measDataCollection.getMeasuredEntityDn();
  String measuredEntitySoftwareVersion = measDataCollection.getMeasuredEntitySoftwareVersion();
  List<MeasInfoList> measInfoLists = measDataCollection.getMeasInfoListList();
  for(MeasInfoList measInfoList:measInfoLists) {
    MeasInfoId measInfoId = measInfoList.getMeasInfoId();
    String sMeasInfoId = measInfoId.getSMeasInfoId();
```

```
        MeasTypes measTypes = measInfoList.getMeasTypes();
        List<String> sMeasTypesList = measTypes.getSMeasTypesListList();
        List<MeasValuesList> measValuesLists = measInfoList.getMeasValuesListList();
        for(MeasValuesList measValuesList:measValuesLists) {
            String measObjInstId = measValuesList.getMeasObjInstId();
            String suspectFlag = measValuesList.getSuspectFlag();
            List<MeasResults> measResults = measValuesList.getMeasResultsList();
            for(MeasResults measResult : measResults) {
              // 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);
              flat.setSMeasType(sMeasType);
              String sValue = measResult.getSValue();
              flat.setSValue(sValue);
              // add the object to the list
              flatList.add(flat);
            }
        }
       }
       return flatList;
      }
}
```

Links

Release H: AVRO

PMProtos.java