Kafka 생태계에서 Protobuf 사용하기

Lee Dongjin | dongjin@apache.org

개요

- Kafka에서 Composite Type 사용하기
  - Kafka Serde: 초간단 설명
  - Avro vs. Protobuf
  - Schema Registry
    - 각종 구현체 소개
- Kafka에 Protobuf 메시지 읽고 쓰기
  - APICurio Registry 사용

Kafka Serde: 초간단 설명 (1)

- Kafka는 기본적으로 (key, value) 의 내용물에 관심이 없음
  - 저장하고자 하는 객체를 byte 뭉치로 변환하는 게 Serializer
  - byte 뭉치를 객체로 변환하는 게 Deserializer
- 기본적으로 제공되는 Serde
  - Primitive types
  - String
  - UUID
  - List

Kafka Serde: 초간단 설명 (2)

- Composite Type를 사용하려면 Custom Serde를 작성해야 함
  - Serializer: Serializer<T> 인터페이스를 구현
  - Deserializer: Deserializer<T> 인터페이스를 구현
- 일단 Serde를 구현해 놓으면 기본적으로 제공되는 것과 동일하게 쓸 수 있다!

```
Properties props = new Properties();
...
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringSerializer");
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    CustomTypeSerializer.class);

Producer<String, CustomType> = new KafkaProducer(props);
```

Kafka Serde: 초간단 설명 (3)

- “그래서 어떤 방법이 있는데?”
  - 대안 1: json 문자열로 직렬화하는 방법
  - 대안 2: 직렬화 Logic을 직접 작성하는 방법
  - 대안 3: 범용 직렬화 라이브러리를 사용하는 방법
    - Avro, Protobuf, Thrift, …
- “그래서 보통 어떻게 쓰나요?”
  - Confluent Schema Registry + Avro가 오랫동안 표준
  - Schema Registry? Avro?

Avro vs. Protobuf (1)

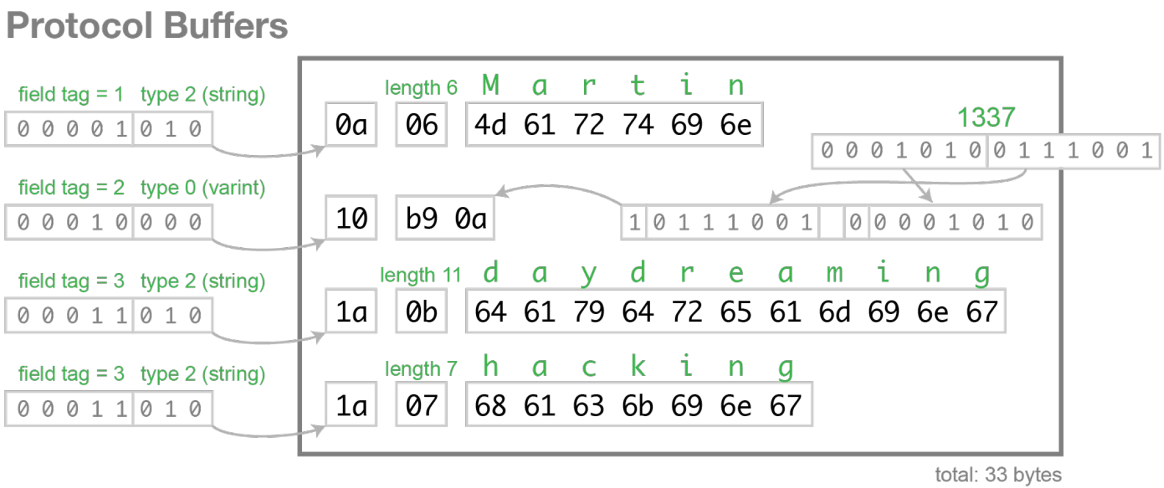
- Protobuf (Google, 2001)
  - Web Service들이 효율적으로 서로 정보를 주고받을 수 있도록 하는 것을 목적으로 개발
  - Schema가 미리 정의되어 있는 상황을 가정하고 만든 시스템
- 고유 문법을 사용
  - 하나의 .proto 파일에 1개 이상의 schema를 정의
    - 깔끔
  - 전용 컴파일러 (protoc)를 사용해서 직렬화/역직렬화 자동 코드 생성
- 동적으로 schema를 정의 + 레코드를 생성(dynamic message creation)할 수 없음

Avro vs. Protobuf (2)

```
message Person {
    required string user_name        = 1;
    optional int64  favourite_number = 2;
    repeated string interests        = 3;
}
```

(Credit: martin kleppman)

Avro vs. Protobuf (3)



(Credit: martin kleppman)

Avro vs. Protobuf (4)

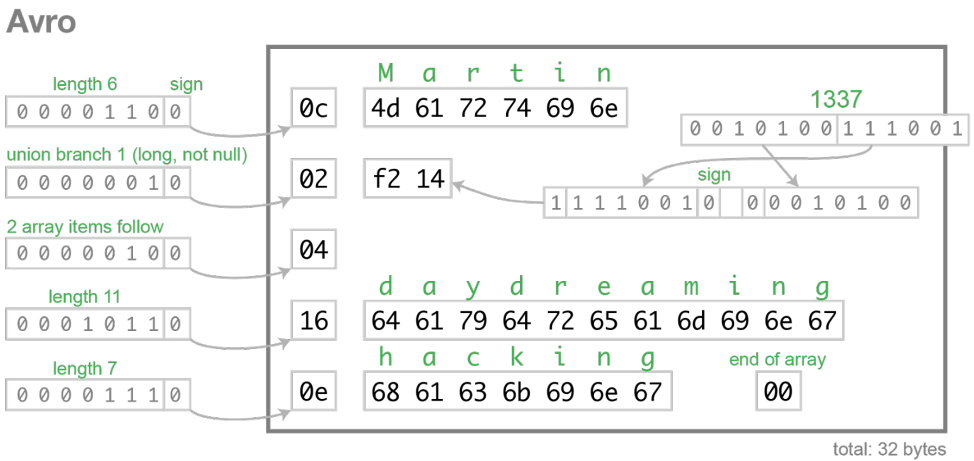
- Avro (Apache Software Foundation, 2009)
  - Hadoop Ecosystem에서 사용하기 위해 개발됨
  - 동적으로 schema를 정의하고, 전달할 수 있어야 함
- json 형식으로 schema를 정의
  - 하나의 json 파일에 하나의 schema
    - 복잡
  - 동적으로 schema를 읽어와서 사용할 수도 있고, 자동 코드 생성을 할 수도 있음
- 동적으로 schema를 정의 + 레코드를 생성(dynamic message creation)할 수 있음

Avro vs. Protobuf (5)

```
{
    "type": "record",
    "name": "Person",
    "fields": [
        {"name": "userName",        "type": "string"},
        {"name": "favouriteNumber", "type": ["null", "long"]},
        {"name": "interests",       "type": {"type": "array", "items":
"string"}}
    ]
}
```

(Credit: martin kleppman)

Avro vs. Protobuf (6)



(Credit: martin kleppman)

Schema Registry (1)

- 남은 문제: Application을 여러 개 유지 보수해야 한다면?
  - Schema가 조금씩 바뀔 때마다 하나씩 일일이 업데이트 해줘야 한다?
- 해법: Http 프로토콜로 접근 가능한, Schema 관리를 위한 전용 서비스
  - Serializer
    - 기존에는 없는 Schema일 경우 자동으로 등록
    - Record 내용물에 (Schema ID, 바이트 뭉치) 형태로 저장
  - Deserializer
    - Record에 저장된 Schema ID 에 해당하는 schema를 읽어와서 바이트 뭉치를 객체로 변환

Schema Registry (2)

- 예: Confluent Schema Registry에서 avro를 사용하는 경우
  - io.confluent:kafka-avro-serializer 사용

```
Properties props = new Properties();
...
props.put("schema.registry.url", "http://localhost:8081");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringSerializer");
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    io.confluent.kafka.serializers.KafkaAvroSerializer.class);

// 동적 Record 사용 (자동 생성된 class를 대신 사용할 수도 있음)
Producer<String, GenericRecord> = new KafkaProducer(props);
```

Schema Registry (3)

- 예: Confluent Schema Registry에서 avro를 사용하는 경우
  - io.confluent:kafka-avro-serializer 사용

```
Properties props = new Properties();
...
props.put("schema.registry.url", "http://localhost:8081");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringDeserializer");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
    io.confluent.kafka.serializers.KafkaAvroDeserializer.class);

// 동적 Record 사용 (자동 생성된 class를 대신 사용할 수도 있음)
Consumer<String, GenericRecord> consumer = new KafkaConsumer<>(props);
```

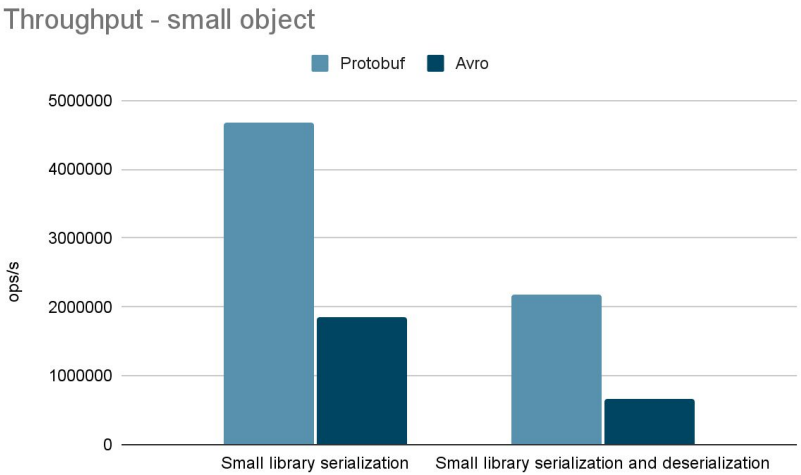
Schema Registry (4)

- Kafka 생태계에서는 오랫동안 Confluent Schema Registry + avro가 표준
- 역사적인 배경
  - Kafka는 원래 Hadoop ecosystem의 일부
  - Linkedin 내부에는 Hadoop 연관 시스템 간에 avro schema를 공유할 수 있는 REST service가 있음
- Confluent Schema Registry
  - 5.5 이전까지 avro만 지원
  - Spark, Flink, Pinot 등 다양한 기술과 연동 기능이 개발되어 있음

Schema Registry (5)

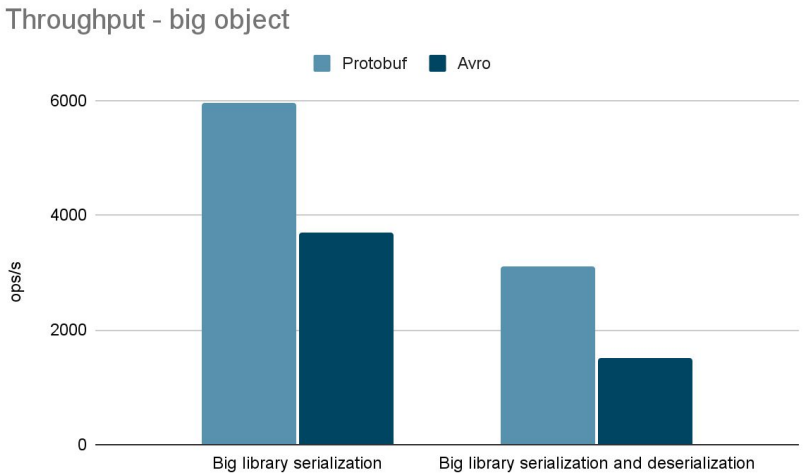
- 문제: protobuf 사용은 사실상 상수
  - gRPC
- Kafka Ecosystem에서의 protobuf 지원
  - Confluent Schema Registry 5.5 이후
- Linkedin이 내부 직렬화 framework를 protobuf로 통일 (2023)
  - 관련글 1
  - 관련글 2
  - Hacker News 관련 논의

Schema Registry (6)



(Credit: softwaremill.com)

Schema Registry (7)



(Credit: softwaremill.com)

Schema Registry - 구현체 (1)

- Confluent Schema Registry
  - Confluent Community License
  - 가장 넓은 사용자 기반 + ecosystem 지원
  - Kafka를 저장소로 사용
  - Json, Avro, Protobuf 지원
- 여기저기서 자체적인 대체품을 내놓기 시작
  - Cloudera, MS Azure, Aiven, Redhat, Buf…
  - 각자 장단점이 있다!

Schema Registry - 구현체 (2)

- Aiven Karapace
  - Apache 2.0 License
  - Json, Avro, Protobuf 지원
  - 장점
    - Schema Registry 6.1.1 호환
    - REST Proxy가 딸려옴

Schema Registry - 구현체 (3)

- Redhat APICurio
  - Apache 2.0 License
  - 장점
    - Debezium과의 연동
    - Json, Avro, Protobuf 외에 GraphQL이나 Kafka Connect Schema, OpenAPI 까지 지원
    - Kafka 이외에 다양한 저장소를 지원 (In-Memory, Postgres, SQL Server, …)
    - 깔끔하고 편리한 web ui
  - 단점
    - 호환성 문제

Kafka에 Protobuf 메시지 읽고 쓰기 (1)

- APICurio - 설치
  - 다운로드 받아서 process 띄우면 끝

```
java \
    -Dapicurio.storage.kind=sql \     # RDBMS(H2)를 저장소로 사용
    -jar app/target/apicurio-registry-app-3.0.0-SNAPSHOT-runner.jar
```

  - 공식 Docker image
  - Helm Chart

Kafka에 Protobuf 메시지 읽고 쓰기 (2)

- io.apicurio:apicurio-registry-serdes-protobuf-serde 사용 (#)

```
Properties props = new Properties();
props.put("apicurio.registry.url",
    "http://localhost:8081/apis/registry/v2");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringSerializer");
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    io.apicurio.registry.serde.protobuf.ProtobufKafkaSerializer.class);

// 자동 생성된 class를 사용
Producer<String, CustomType> = new KafkaProducer(props);
```

Kafka에 Protobuf 메시지 읽고 쓰기 (3)

- io.apicurio:apicurio-registry-serdes-protobuf-serde 사용 (#)

```
Properties props = new Properties();
props.put("apicurio.registry.url",
    "http://localhost:8081/apis/registry/v2");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringDeserializer");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
  io.apicurio.registry.serde.protobuf.ProtobufKafkaDeserializer.class);

// 동적 Record 사용 (자동 생성된 class를 대신 사용할 수도 있음)
Consumer<String, DynamicMessage> consumer = new KafkaConsumer<>(props);
```

질문 받습니다.