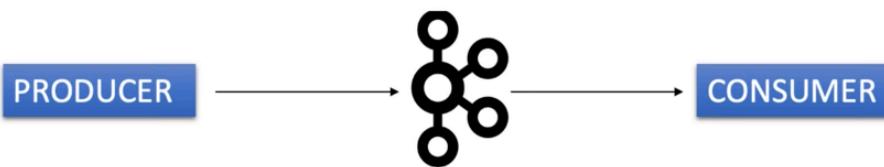


KAFKA – AVRO SCHEMA REGISTRY

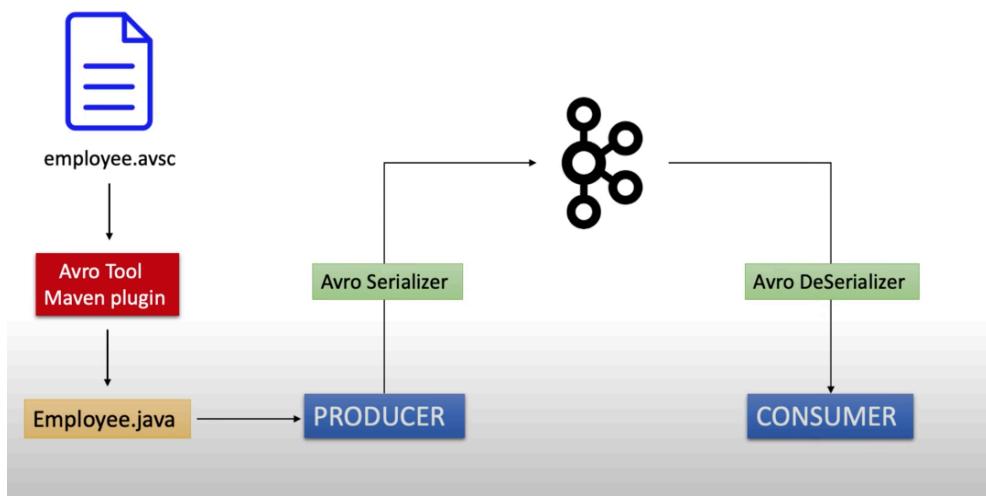
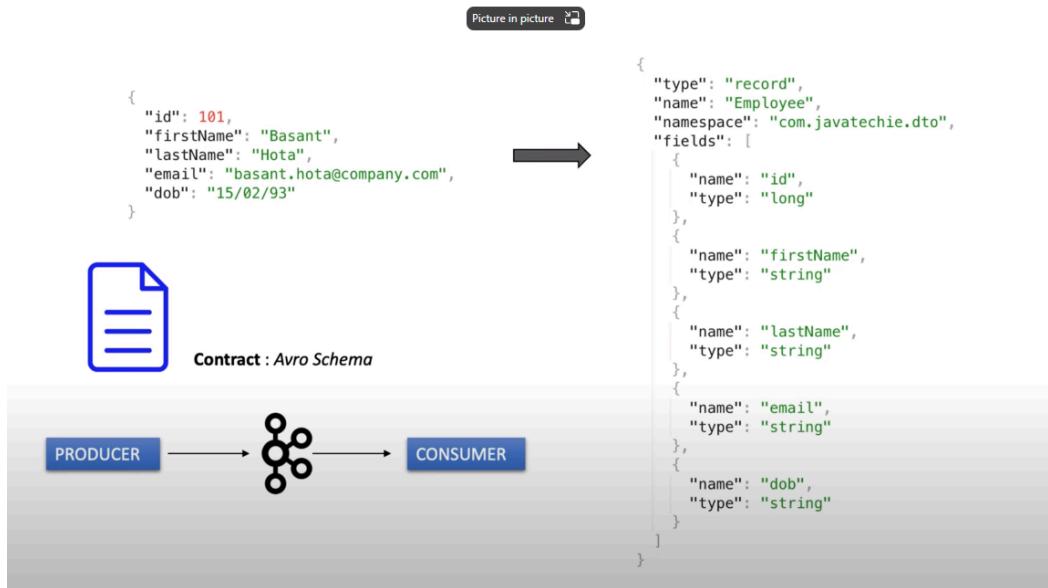
Change in the contract

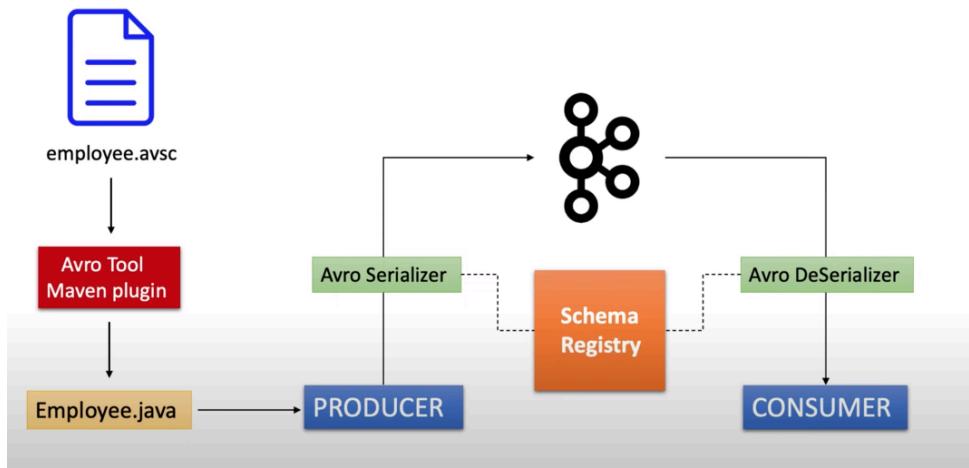
```
{  
  "id": 101,  
  "firstName": "Basant",  
  "lastName": "Hota",  
  "email": "basant.hota@company.com",  
  "dob": "15/02/93"  
}
```



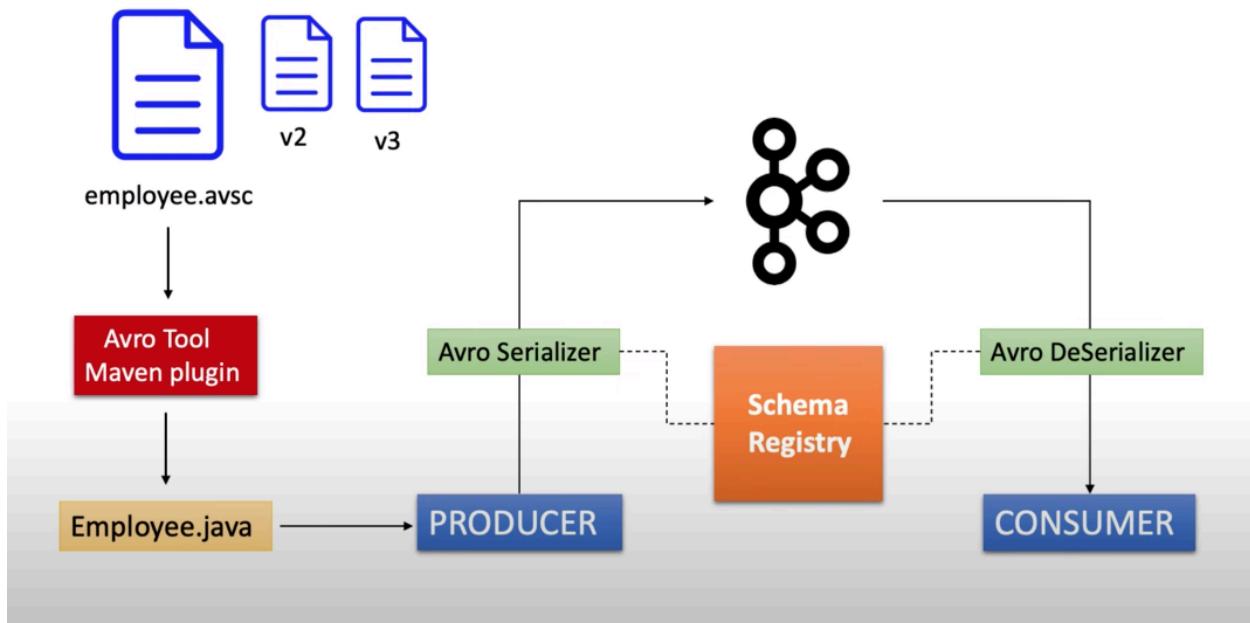
```
{  
  "id": 101,  
  "firstName": "Basant",  
  "middleName": "kumar",  
  "lastName": "Hota",  
  "emailId": "basant.hota@company.com"  
}
```







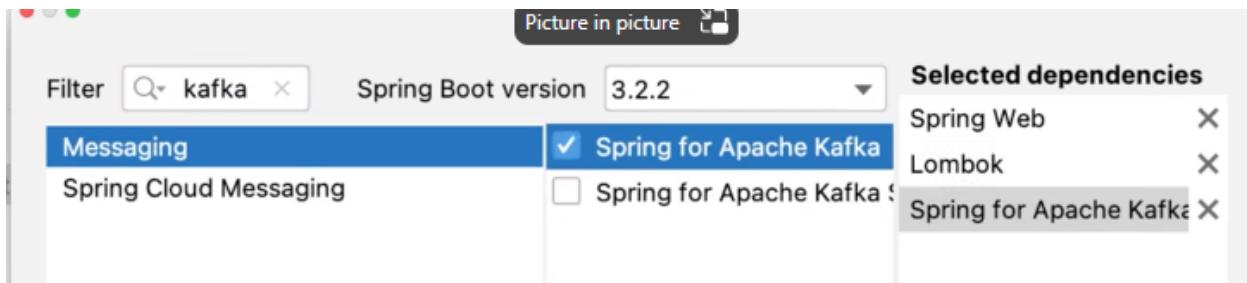
The purpose of schema registry is to store , retrieve and evaluate schema in consist manner.



New changes will come to the schema over a period of time. That time we have to check forward and backward compatibility.

This is what will be explained in this document.

Create a project



Docker compose

```
version: "3"
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.4.0
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
```

The screenshot shows the IntelliJ IDEA interface with the 'docker-compose.yml' file open in the editor. The code defines a single service named 'zookeeper' using the 'confluentinc/cp-zookeeper:5.4.0' image. It maps port 2181 to 2181 and sets environment variables ZOOKEEPER_CLIENT_PORT and ZOOKEEPER_TICK_TIME. The project structure on the left shows files like .idea, .mvn, src, target, .gitignore, and pom.xml.

[spring-kafka-avro/docker-compose.yml at main · Java-Techie-jt/spring-kafka-avro · GitHub](https://github.com/Java-Techie-jt/spring-kafka-avro)

spring-kafka-avro / docker-compose.yml

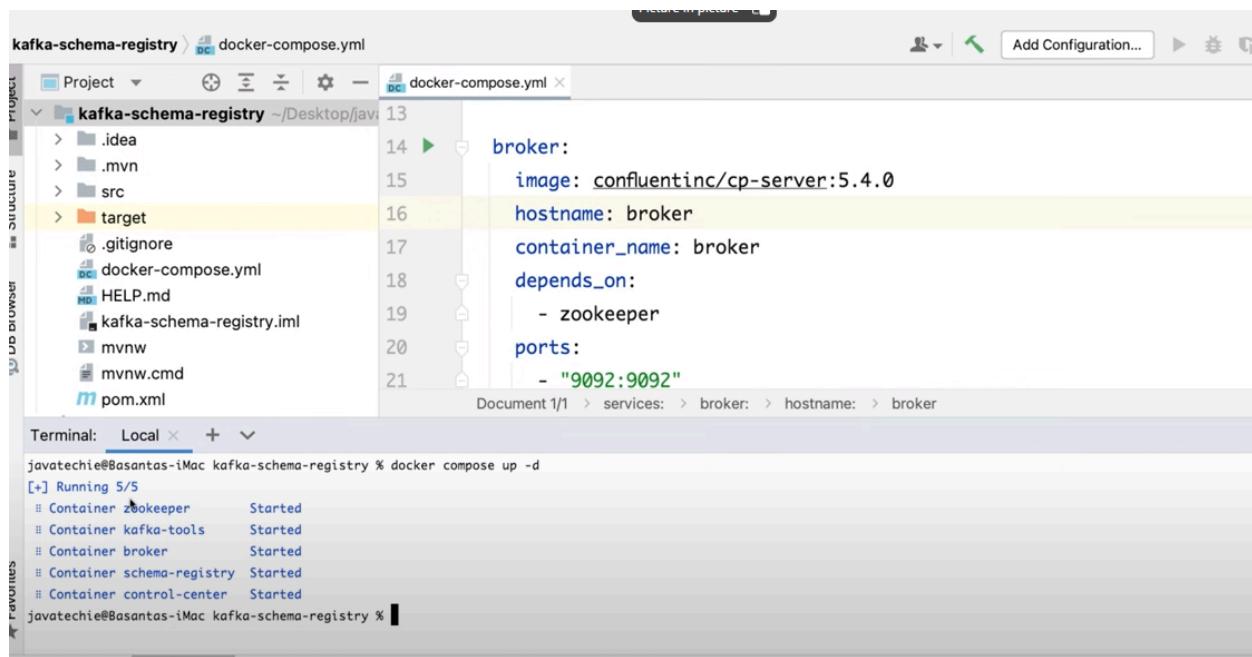
Code Blame 75 lines (70 loc) • 2.33 KB

```
3 services:
4   zookeeper:
5     image: confluentinc/cp-zookeeper:5.4.0
6     hostname: zookeeper
7     container_name: zookeeper
8     ports:
9       - "2181:2181"
10    environment:
11      ZOOKEEPER_CLIENT_PORT: 2181
12      ZOOKEEPER_TICK_TIME: 2000
13
14  broker:
15    image: confluentinc/cp-server:5.4.0
16    hostname: broker
17    container_name: broker
18    depends_on:
19      - zookeeper
20    ports:
21      - "9092:9092"
22    environment:
23      KAFKA_BROKER_ID: 1
24      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
25      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
26      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:9092,PLAINTEXT_HOST://localhost:9092
27      KAFKA_METRIC_REPORTERS: io.confluent.metrics.reporter.ConfluentMetricsReporter
28      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
29      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
30      KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 1
31      CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: broker:9092
32      CONFLUENT_METRICS_REPORTER_ZOOKEEPER_CONNECT: zookeeper:2181
33      CONFLUENT_METRICS_REPORTER_TOPIC_REPLICAS: 1
34      CONFLUENT_METRICS_ENABLE: "true"
35      CONFLUENT_SUPPORT_CUSTOMER_ID: "anonymous"
36
37  kafka-tools:
38    image: confluentinc/cp-kafka:5.4.0
39    hostname: kafka-tools
40    container_name: kafka-tools
41    command: ["tail", "-f", "/dev/null"]
42    network_mode: "host"
43
44  schema-registry:
45    image: confluentinc/cp-schema-registry:5.4.0
46    hostname: schema-registry
47    container_name: schema-registry
48    depends_on:
49      - zookeeper
50      - broker
```

```

37     kafka-tools:
38         image: confluentinc/cp-kafka:5.4.0
39         hostname: kafka-tools
40         container_name: kafka-tools
41         command: ["tail", "-f", "/dev/null"]
42         network_mode: "host"
43
44     schema-registry:
45         image: confluentinc/cp-schema-registry:5.4.0
46         hostname: schema-registry
47         container_name: schema-registry
48         depends_on:
49             - zookeeper
50             - broker
51         ports:
52             - "8081:8081"
53         environment:
54             SCHEMA_REGISTRY_HOST_NAME: schema-registry
55             SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: "zookeeper:2181"
56
57     control-center:
58         image: confluentinc/cp-enterprise-control-center:5.4.0
59         hostname: control-center
60         container_name: control-center
61         depends_on:
62             - zookeeper
63             - broker
64             - schema-registry
65         ports:
66             - "9021:9021"
67         environment:
68             CONTROL_CENTER_BOOTSTRAP_SERVERS: 'broker:29092'
69             CONTROL_CENTER_ZOOKEEPER_CONNECT: 'zookeeper:2181'
70             CONTROL_CENTER_SCHEMA_REGISTRY_URL: "http://schema-registry:8081"
71             CONTROL_CENTER_REPLICATION_FACTOR: 1
72             CONTROL_CENTER_INTERNAL_TOPICS_PARTITIONS: 1
73             CONTROL_CENTER_MONITORING_INTERCEPTOR_TOPIC_PARTITIONS: 1
74             CONFLUENT_METRICS_TOPIC_REPLICATION: 1
75             PORT: 9021

```



Controller center ui

The screenshot shows the Confluent Controller Center UI. At the top, there's a navigation bar with tabs like 'localhost:9023/clusters' and 'Control Center'. Below the navigation is a dark header bar with the 'Confluent' logo. The main content area is titled 'Clusters' and shows a summary: '1 Healthy clusters' and '0 Unhealthy clusters'. A search bar labeled 'Search cluster name' is present. Below this, a card displays the details for the 'controlcenter.cluster': it is 'Running' with 1 Broker, 101 Partitions, 40 Topics, and production/consumption rates of 9.08kB/s and 6.79kB/s respectively. It also lists 'Connected services' with 0 KSQL clusters and 0 Connect clusters.

Step 1: Schema definition

The screenshot shows an IntelliJ IDEA interface with the 'kafka-schema-registry' project open. The 'employee.avsc' file is selected in the code editor. The code editor shows the following Avro schema:

```
2 "namespace": "com.javatechie.dto",
3 "type": "record",
4 "name": "Employee",
5 "fields": [
6     {
7         "name": "id",
8         "type": "string"
9     },
10    {
11        "name": "firstName",
12        "type": "string"
13    }
14 ]
```

The project structure on the left shows the directory tree: 'src/main/resources/avro/employee.avsc'. Other files like 'docker-compose.yml', 'application.properties', and 'static/templates' are also visible.

Step 2: Add Avro plugin

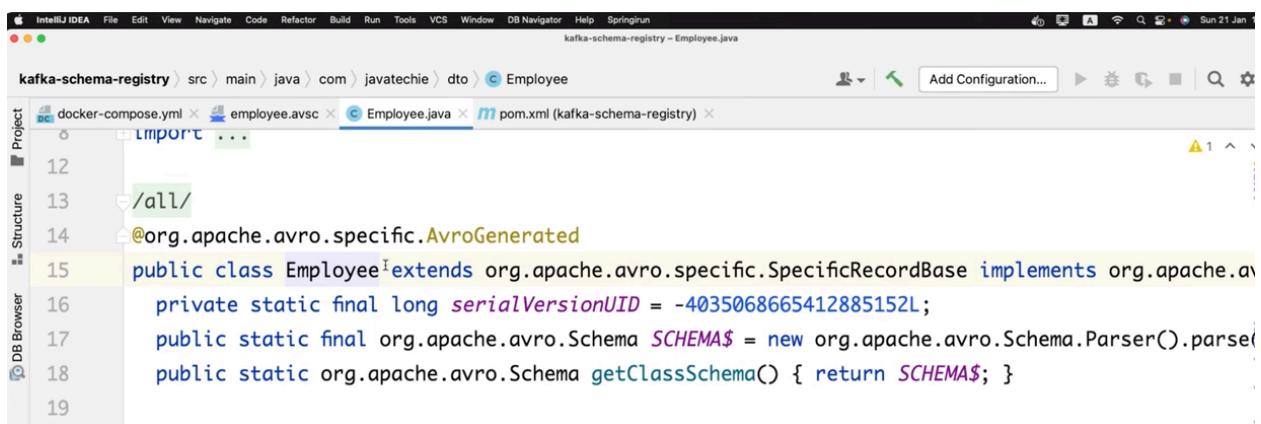
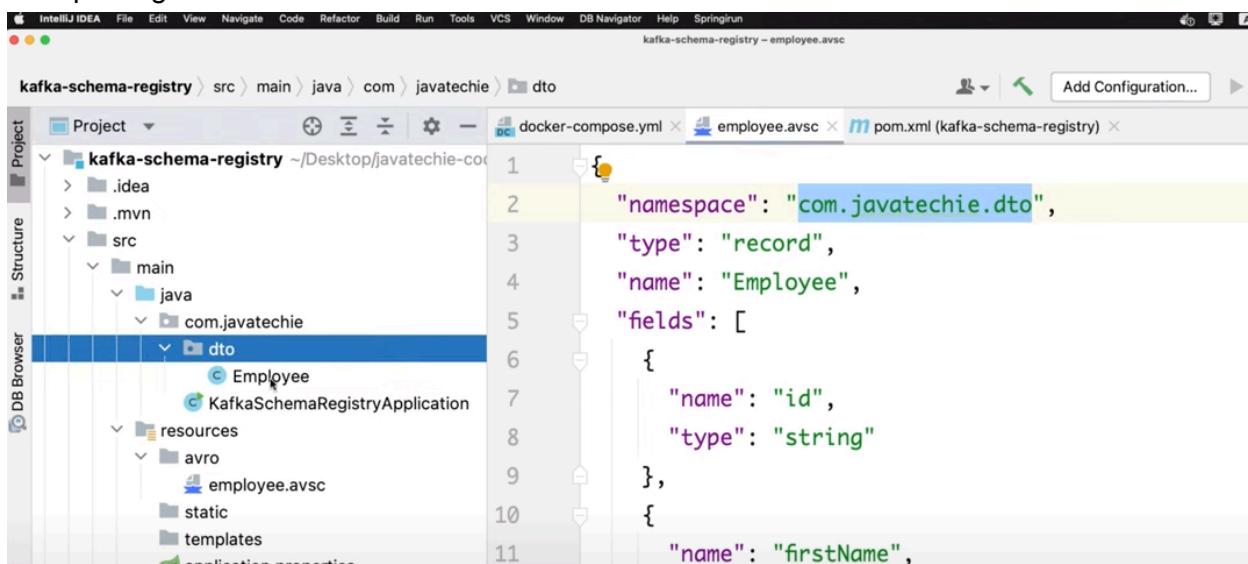
```
85             </configuration>
86         </plugin>
87     </plugins>
88     <groupId>org.apache.avro</groupId>
89     <artifactId>avro-maven-plugin</artifactId>
90     <version>1.8.2</version>
91     <executions>
92         <execution>
93             <id>schemas</id>
94             <phase>generate-sources</phase>
95             <goals>
96                 <goal>schema</goal>
97             </goals>
98             <configuration>
99                 <sourceDirectory>${project.basedir}/src/main/resources/</sourceDirectory>
100                <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
101            </configuration>
102        </execution>
103    </executions>
104  </plugin>
105 </plugins>
106 </build>
```

Here we defined what is the source and destination.

Step 3: simply run install which will create dto package and employee.java



Java package and class will be created as defined in schema



Step 4: Create producer



Create a topic in configuration class as shown below.

```
Code Blame 18 lines (14 loc) · 462 Bytes

1 package com.javatechie.config;
2
3 import org.apache.kafka.clients.admin.NewTopic;
4 import org.springframework.beans.factory.annotation.Value;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 @Configuration
9 public class KafkaConfig {
10
11     @Value("${topic.name}")
12     private String topicName;
13
14     @Bean
15     public NewTopic createTopic(){
16         return new NewTopic(topicName, 3, (short) 1);
17     }
18 }
```

Use this topic in producer as shown below

Code

Blame

```
1 package com.javatechie.producer;
2
3 import com.javatechie.dto.Employee;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.kafka.core.KafkaTemplate;
7 import org.springframework.kafka.support.SendResult;
8 import org.springframework.stereotype.Service;
9
10 import java.util.UUID;
11 import java.util.concurrent.CompletableFuture;
12
13 @Service
14 public class KafkaAvroProducer {
15
16     @Value("${topic.name}")
17     private String topicName;
18
19     @Autowired
20     private KafkaTemplate<String, Employee> template;
21
22
23     public void send(Employee employee){
24         CompletableFuture<SendResult<String, Employee>> future = template.send(topicName, UUID.randomUUID().toString(),employee);
25         future.whenComplete((result, ex) -> {
26             if (ex == null) {
27                 System.out.println("Sent message=[ " + employee +
28                     "] with offset=[ " + result.getRecordMetadata().offset() + "]");
29             } else {
30                 System.out.println("Unable to send message=[ " +
31                     employee + "] due to : " + ex.getMessage());
32             }
33         });
34     }
35 }
```

Step 5: Create consumer as shown below

```
1 package com.javatechie.consumer;
2
3 import com.javatechie.dto.Employee;
4 import lombok.extern.slf4j.Slf4j;
5 import org.apache.kafka.clients.consumer.ConsumerRecord;
6 import org.springframework.kafka.annotation.KafkaListener;
7 import org.springframework.stereotype.Service;
8
9 @Service
10 @Slf4j
11 public class KafkaAvroConsumer {
12
13     @KafkaListener(topics = "${topic.name}")
14     public void read(ConsumerRecord<String, Employee> consumerRecord) {
15         String key = consumerRecord.key();
16         Employee employee = consumerRecord.value();
17         log.info("Avro message received for key : " + key + " value : " + employee.toString());
18
19     }
20 }
```

Create port for this app

```

Project employee.avsc Employee.java KafkaAvroProducer.java KafkaA
2   name: javatechie-avro
3
4   server:
5     port: 8181
6

```

Step 6: Create producer Avro serializer and its schema registry address.

```

compose.yml employee.avsc Employee.java KafkaAvroProducer.java KafkaAvroConsumer.java EventController.java KafkaConfig.java application.yml
8   spring:
9     kafka:
10       producer:
11         bootstrap-servers: "127.0.0.1:9092"
12         key-serializer: "org.apache.kafka.common.serialization.StringSerializer"
13         value-serializer: "io.confluent.kafka.serializers.KafkaAvroSerializer"
14       properties:
15         schema:
16           registry:
17             url : http://127.0.0.1:8081

```

Step 7: Create consumer Avro de-serializer and its schema registry address as shown below.

```

src
  main
    java/com/javatechie
      config
        KafkaConfig.java
      consumer
        KafkaAvroConsumer.java
      controller
        EventController.java
      dto
      producer
        KafkaAvroProducer.java
        KafkaSchemaRegistryApplica...
      resources
      avro
        application.properties
      application.yml
    test
    README.md
    docker-compose.yml
    pom.xml
topic:
  name: javatechie-avro
  server:
    port: 8181
  spring:
    kafka:
      bootstrap-servers: "127.0.0.1:9092"
      producer:
        keySerializer: "org.apache.kafka.common.serialization.StringSerializer"
        valueSerializer: "io.confluent.kafka.serializers.KafkaAvroSerializer"
      properties:
        schema:
          registry:
            url: "http://127.0.0.1:8081"
      consumer:
        group-id: "javatechie-new"
        keyDeserializer: "org.apache.kafka.common.serialization.StringDeserializer"
        valueDeserializer: "io.confluent.kafka.serializers.KafkaAvroDeserializer"
        autoOffsetReset: "earliest"
      properties:
        schema:
          registry:
            url: "http://127.0.0.1:8081"
        specific:
          avro:
            reader: "true"

```

Now All the steps are completed let us start the application and test it

```

25 registry:
Document 1/1 > spring: > kafka: > consumer: > group-id: > javatechie-new
Run: KafkaSchemaRegistryApplication
bscriptionState : [Consumer clientId=consumer-javatechie-new-1, groupId=javatechie-new] Resetting offset for partition javatechie-new-0
bscriptionState : [Consumer clientId=consumer-javatechie-new-1, groupId=javatechie-new] Resetting offset for partition javatechie-new-1
bscriptionState : [Consumer clientId=consumer-javatechie-new-1, groupId=javatechie-new] Resetting offset for partition javatechie-new-2
stenerContainer : javatechie-new: partitions assigned: [javatechie-avro-0, javatechie-avro-1, javatechie-avro-2]
yApplication : Started KafkaSchemaRegistryApplication in 5.642 seconds (process running for 6.652)

Run Problems Terminal Jenkins Linter TODO Services Build Dependencies Event Log
Build completed successfully in 2 sec, 65 ms (moments ago) 274:1 LF UTF-8 2 spaces No JSON schema

```

Topic with two partitions was created.

Topics	Availability	Throughput
Topic name	Under replicated partitions	Bytes/sec produced
javatechie-avro	0 of 3	-
	0 of 3	-
	0 of 0	-

At the beginning we may not see schema in the control-center. Once we publish it will register.

A message value schema is not set
Set a schema to ensure the compatibility of your data and code.
Set a schema

Not even in schema registry also we won't see

```

{"value":[]}

```

Publish a message from postman

```

POST http://localhost:8181/events
Body
1 {
2   "id": "DHJKWH033",
3   "firstName": "Basant",
4   "lastName": "Hota",
5   "email": "basant.hota@gmail.com",
6   "dob": "15/04/92",
7   "age": 30
8 }

```

Response: 200 OK 1339 ms 183 B

```

1 [message published]

```

```

2024-01-21T14:15:18.428+05:30 INFO 29874 --- [producer-1] org.apache.kafka.clients.Metadata : [Producer clientId=producer-1j clusterId: XINIZK-QARAUU/OXI_GKw
2024-01-21T14:15:18.428+05:30 INFO 29874 --- [ad l producer-1] o.a.k.c.p.internals.TransactionManager : [Producer clientId=producer-1] ProducerId set to 4000 with epoch 0
Sent message=[{"id": "DHJKWH033", "firstName": "Basant", "lastName": "Hota", "email": "basant.hota@gmail.com", "dob": "15/04/92", "age": 30}] with offset=[0]
2024-01-21T14:15:19.982+05:30 INFO 29874 --- [ntainer#0-0-C-1] c.javatechie.consumer.KafkaAvroConsumer : Avro message received for key : 4b865fa3-8652-4425-8e29-867e7dccb548 val

```

Now we can see schema

Value

```

1 {
2   "fields": [
3     {
4       "name": "id",
5       "type": "string"
6     },
7     {
8       "name": "firstName",
9       "type": "string"
10    },
11    {
12      "name": "lastName",
13      "type": "string"
14    },
15    {
16      "default": "",
17      "name": "email",
18      "type": "string"
19    },
20    {
21      "name": "dob"
22    }
23  ]
24}

```

In schema registry, The below endpoint will return topic

```

[{"name": "javatechie-avro-value"}
]

```

The below endpoint will return avro schema

```

{
  "subject": "javatechie-avro-value",
  "version": 1,
  "id": 61,
  "schema": "{\"type\":\"record\", \"name\":\"Employee\", \"namespace\":\"com.javatechie.dto\", \"fields\":[{\"name\":\"id\", \"type\":\"string\"}, {\"name\":\"email\", \"type\":\"string\"}, {\"default\":\"\"}, {\"name\":\"dob\", \"type\":\"string\"}, {\"name\":\"age\", \"type\":\"int\"}]}"
}

```

Version of the schema is 1.

Now modify the schema remove DOB and age and change email to emailId



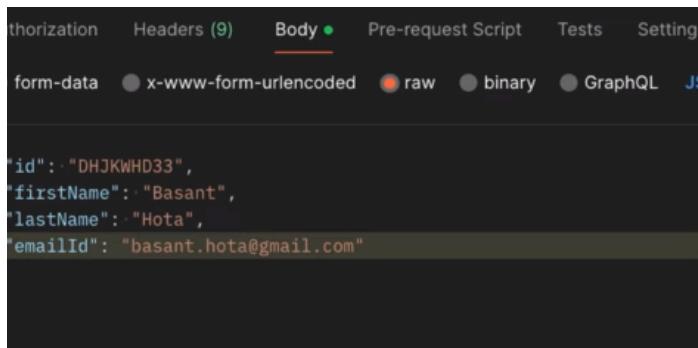
```
5     "fields": [
6         {
7             "name": "id",
8             "type": "string"
9         },
10        {
11            "name": "firstName",
12            "type": "string"
13        },
14        {
15            "name": "lastName",
16            "type": "string"
17        },
18        {
19            "name": "emailId",
20            "type": "string",
21            "default": ""
22        }
23    }
24 }
```

Regenerate the avro



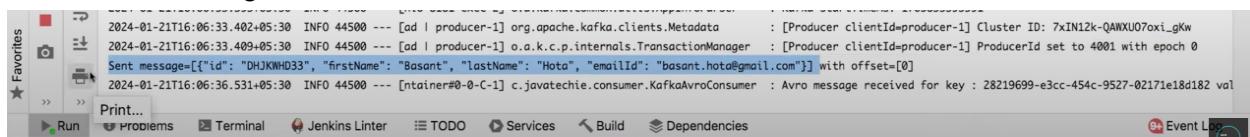
```
46
47     /**
48      * Deserializes a Employee From a ByteBuffer. */
49     public static Employee fromByteBuffer(
50         java.nio.ByteBuffer b) throws java.io.IOException {
51         return DECODER.decode(b);
52     }
53
54     @Deprecated public java.lang.CharSequence id;
55     @Deprecated public java.lang.CharSequence firstName;
56     @Deprecated public java.lang.CharSequence lastName;
57     @Deprecated public java.lang.CharSequence emailId;
58 }
```

start the application



```
"id": "DHJKWHD33",
"firstName": "Basant",
"lastName": "Hota",
"emailId": "basant.hota@gmail.com"
```

Publish the message



```
2024-01-21T16:06:33.402+05:30 INFO 44500 --- [ad | producer-1] org.apache.kafka.clients.Metadata : [Producer clientId=producer-1] Cluster ID: 7xIN12k-QAWXU07oxi_gKw
2024-01-21T16:06:33.409+05:30 INFO 44500 --- [ad | producer-1] o.a.k.c.p.internals.TransactionManager : [Producer clientId=producer-1] ProducerId set to 4001 with epoch 0
2024-01-21T16:06:36.531+05:30 INFO 44500 --- [Container#0-C-1] c.javatechie.consumer.KafkaAvroConsumer : Avro message received for key : 28219699-e3cc-454c-9527-02171e18d182 val
```

We can see the comparison between previous and current versions in the control center as shown below.

The screenshot shows the Confluent Schema Registry interface for a topic named 'CONTROLCENTER.CLUST...'. The left sidebar has a 'Topics' section selected. The main area has tabs for 'Overview', 'Messages', 'Schema' (which is selected), and 'Configuration'. A checkbox labeled 'Turn on version diff' is checked. Below it, two dropdown menus show 'version 1' and 'version 2 (current)'. The schema code is displayed in two panes:

```
version 1
```

```
version 2 (current)
```

```
1 { "fields": [ 2 ( 3 { "name": "id", 4 "type": "string" 5 }, 6 { 7 ( "name": "firstName", 8 "type": "string" 9 ), 10 { 11 ( "name": "lastName", 12 "type": "string" 13 ), 14 { 15 ( 16 { "default": "", 17 "name": "email", 18 "type": "string" 19 ), 20 { 21 { "name": "dob", 22 "type": "string" 23 }, 24 { 25 { "name": "age", 26 "type": "int" 27 ) } ] }
```

```
1 { "fields": [ 2 ( 3 { "name": "id", 4 "type": "string" 5 }, 6 { 7 ( "name": "firstName", 8 "type": "string" 9 ), 10 { 11 ( "name": "lastName", 12 "type": "string" 13 ), 14 { 15 ( 16 { "default": "", 17 "name": "emailId", 18 "type": "string" 19 ) } ] }
```

In the schema registry we see a new version.

```
localhost:8081/subjects/javatechie-avro-value/versions/latest
fav-songs Interview AWS softwares spring boot EMOJI Live Classes next-video-poc CMD medium BBMP Teachcode Courses reference devops-ref
{
    "subject": "javatechie-avro-value",
    "version": 2,
    "id": 62,
    "schema": "{\"type\":\"record\",\"name\":\"Employee\",\"namespace\":\"com.javatechie.dto\",\"fields\":[{\"name\":\"id\",\"type\":\"string\"},{\"name\":\"emailId\",\"type\":\"string\",\"default\":\"\"}]}"
}
```

It maintain schema history

The screenshot shows the Confluent Control Center interface for a cluster named 'CONTROLCENTER.CLUST...'. The 'Topics' tab is selected. On the right, the 'Schema' tab is active, showing the schema for the topic 'avatechile-avro'. The schema is defined as follows:

```
version 1
version 2 (current)
version 1
{
  "fields": [
    {
      "name": "id",
      "type": "string"
    }
  ]
}
```

Add middle name and mayen install

The screenshot shows the IntelliJ IDEA interface with several tabs at the top: Application.java, application.yml, employee.avsc, and Maven. The Maven tool window is open on the right, showing the 'Lifecycle' section with 'install' highlighted. Below the tool windows, the terminal window displays the build output:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 17.948 s
[INFO] Finished at: 2024-01-21T16:12:00+05:30
[INFO] -----
Process finished with exit code 0
```

Middle name is added in java file.

The screenshot shows the IntelliJ IDEA interface with several tabs at the top: KafkaSchemaRegistryApplication.java, application.yml, employee.avsc, Employee.java, and EventController.java. The Employee.java editor is active, showing the following code:

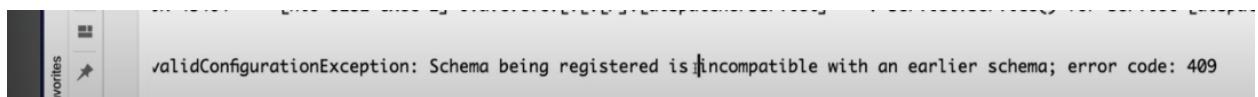
```
51     return DECODER.decode(b);
52 }
53
54 @Deprecated public java.lang.CharSequence id;
55 @Deprecated public java.lang.CharSequence firstName;
56 @Deprecated public java.lang.CharSequence middleName; // Warning icon here
57 @Deprecated public java.lang.CharSequence lastName;
58 @Deprecated public java.lang.CharSequence emailId;
```

We got error with this new change

The screenshot shows the Postman interface. A POST request is being made to `http://localhost:8181/events`. The request body is a JSON object:

```
1 {
2   "id": "DHJKWHD33",
3   "firstName": "Basant",
4   "middleName": "kumar",
5   "lastName": "Hota",
6   "emailId": "basant.hota@gmail.com"
7 }
```

The response status is 500 Internal Server Error, with a timestamp of 2024-01-21T10:43:22.154+00:00, a status of 500, an error message of "Internal Server Error", and a path of "/events".



For any new field mark it as default as shown below.

The screenshot shows a code editor with the file `employee.avsc` open. The schema defines three fields:

```
1 {
2   "name": "firstName",
3   "type": "string"
4 },
5 {
6   "name": "middleName",
7   "type": "string",
8   "default": ""
9 },
10 {
11   "name": "lastName",
12   "type": "string"
13 }
```

The line `"default": ""` is highlighted with a blue selection bar.

Rebuild the project and run

POST http://localhost:8181/events

Params Authorization Headers (9) Body **Body** Pre-request Script Tests Settings Cookies Beautify

```

1 {
2   "id": "DJKWHD33",
3   "firstName": "Basant",
4   "middleName": "kumar",
5   "lastName": "Hota",
6   "emailId": "basant.hota@gmail.com"
7 }
  
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text ↗

1 message published !

200 OK 870 ms 183 B Save as example

ALL TOPICS >
javatechie-avro

Overview Messages Schema Configuration

Cleanup policy Delete

Partitions 3

Bytes in/sec 0

Bytes out/sec

timestampType	headers	key	value.id	value.firstName	value.middleName	value.lastName
CREATE_TIME	[]	d03dae15-273d-40...	DJKWHD33	Basant	kumar	Hota

ALL TOPICS > JAVATECHIE-AVRO > SCHEMA > VALUE >

Version history

Overview Messages Schema Configuration

Turn on version diff

version 3 (current)

version 3 (current)

version 2

version 1

```

3   {
4     "name": "id",
5     "type": "string"
6   },
7   {
  
```

Version one is Fresh avro and 2 is modification and 3 is addition.