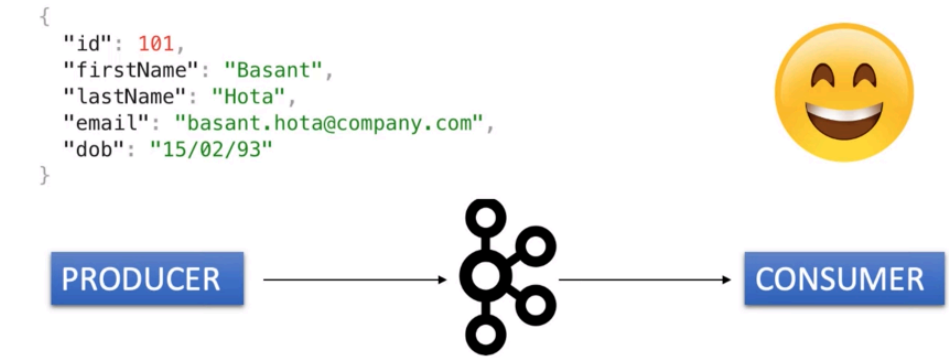
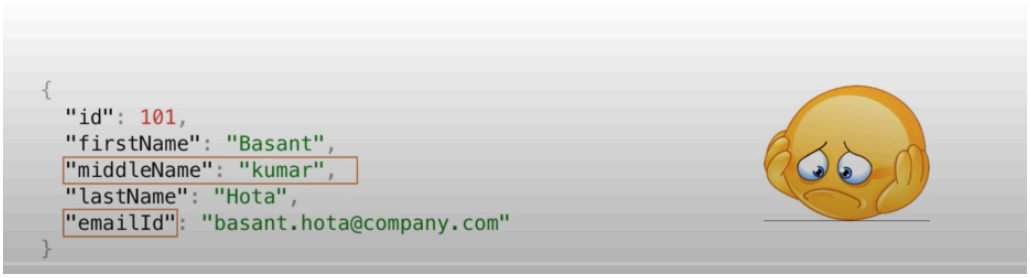
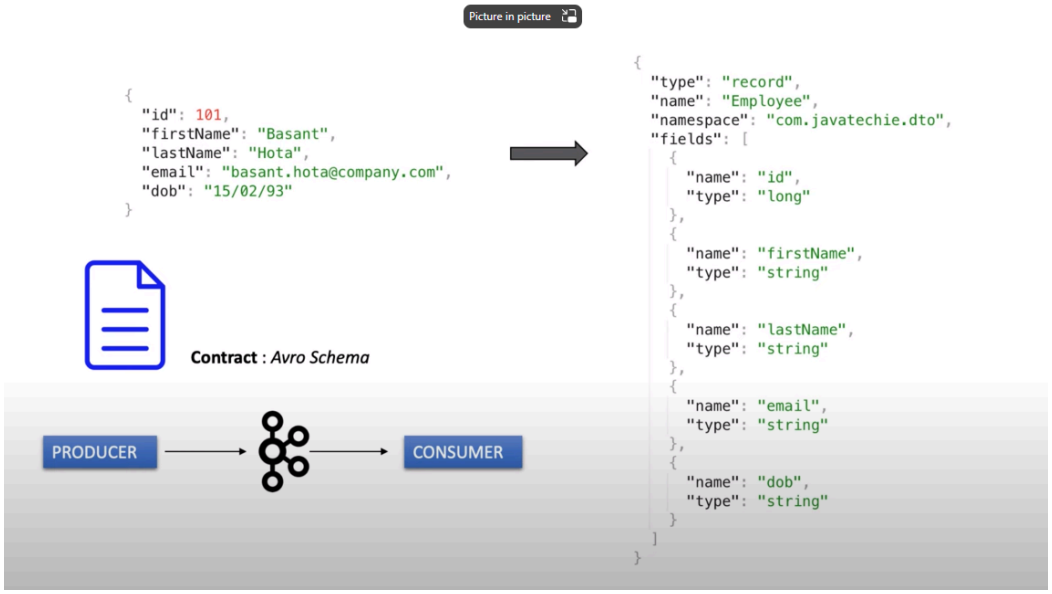
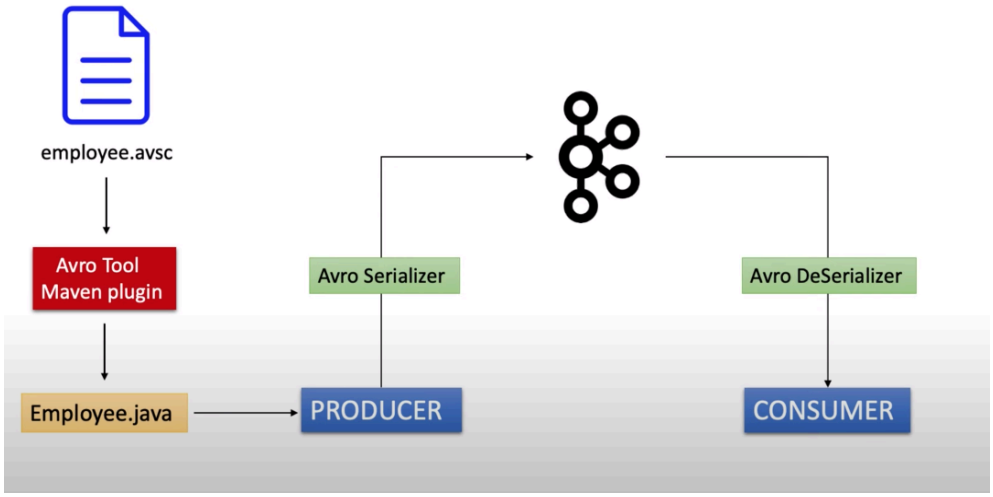
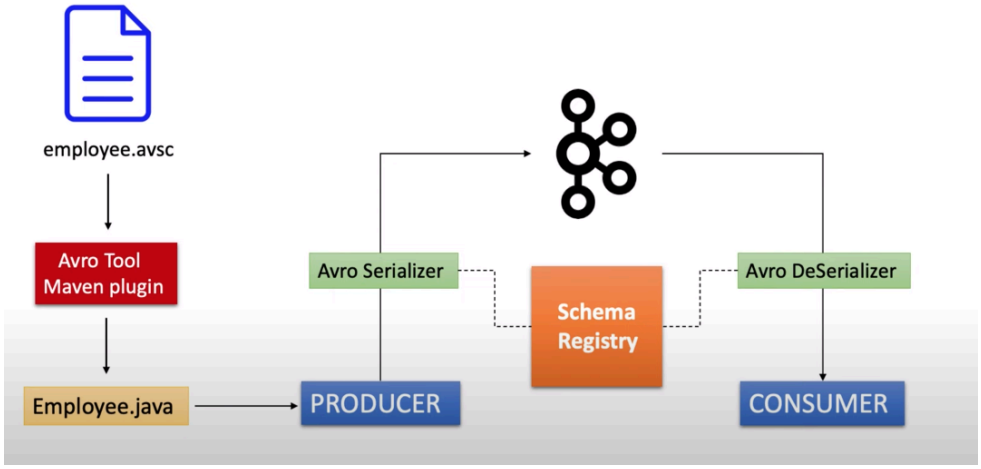
Change in the contract



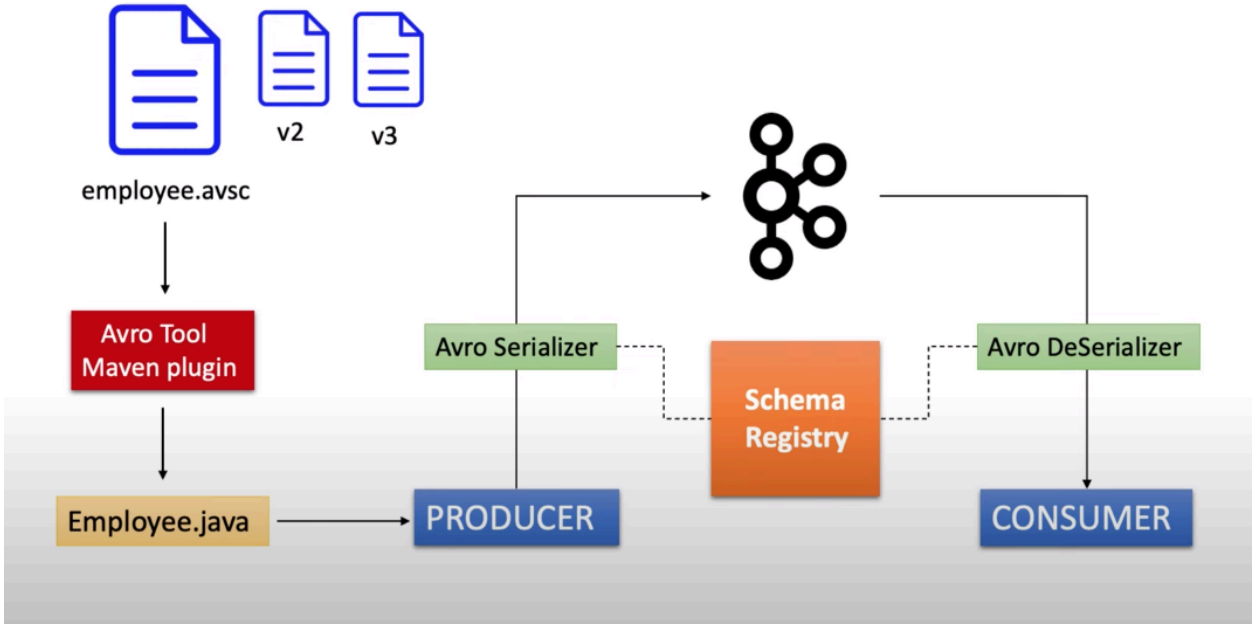



Picture in picture
{
"id": 101,
"firstName": "Basant",
"lastName": "Hota",
"email": "basant.hota@company.com",
"dob": "15/02/93"
}
{
"type": "record",
"name": "Employee",
"namespace": "com.javatechie.dto",
"fields": [
{
"name": "id",
"type": "long"
},
{
"name": "firstName",
"type": "string"
},
{
"name": "lastName",
"type": "string"
},
{
"name": "email",
"type": "string"
},
{
"name": "dob",
"type": "string"
}
]
}
Contract : Avro Schema
PRODUCER
CONSUMER


employee.avsc
Avro Tool
Maven plugin
Employee.java
Avro Serializer
PRODUCER
Avro DeSerializer
CONSUMER



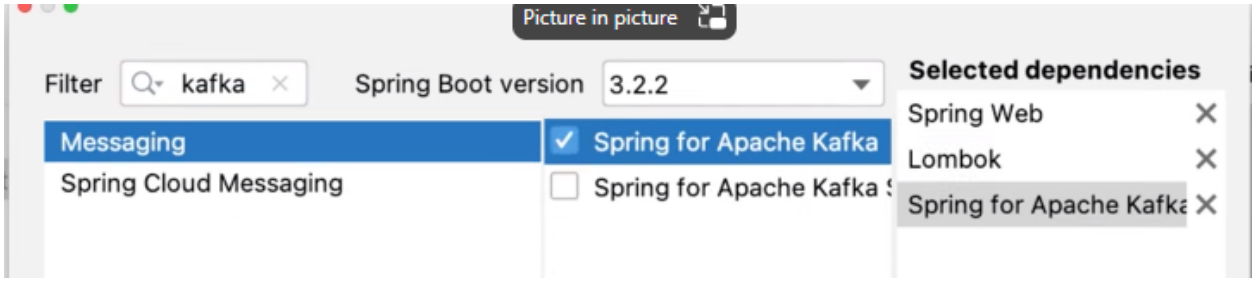
The purpose of schema registry is to store , retrieve and evaluate schema in consist manner.



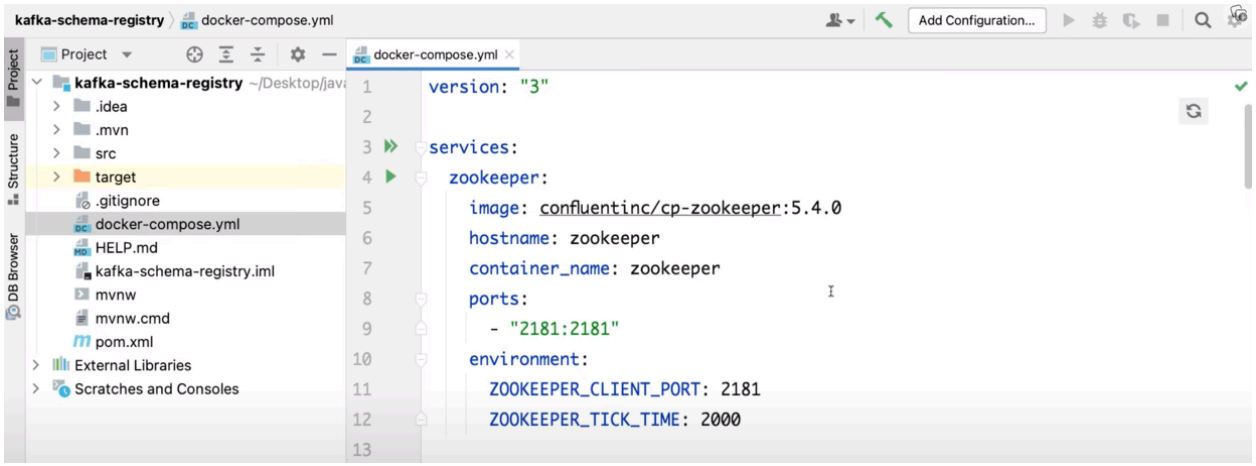
New changes will come to the schema over a period of time. That time we have to check forward and backward compatibility.

This is what will be explained in this document.

Create a project



Docker compose



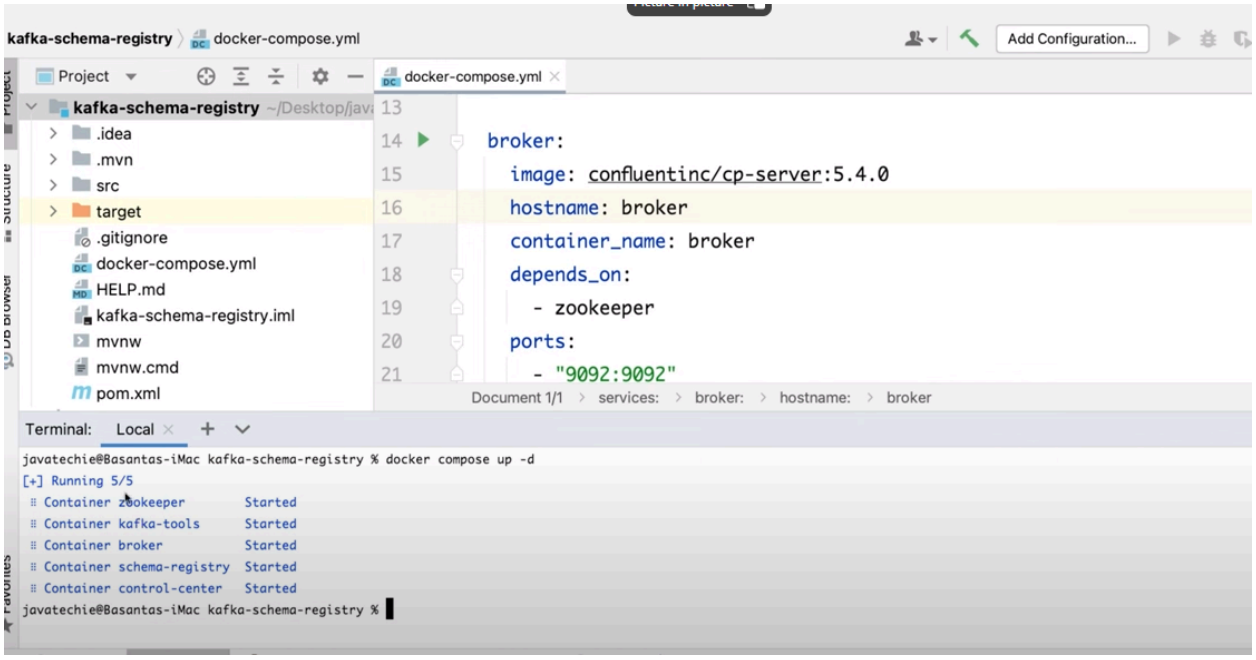
[spring-kafka-avro/docker-compose.yml at main · Java-Techie-jt/spring-kafka-avro · GitHub]

spring-kafka-avro / docker-compose.yml

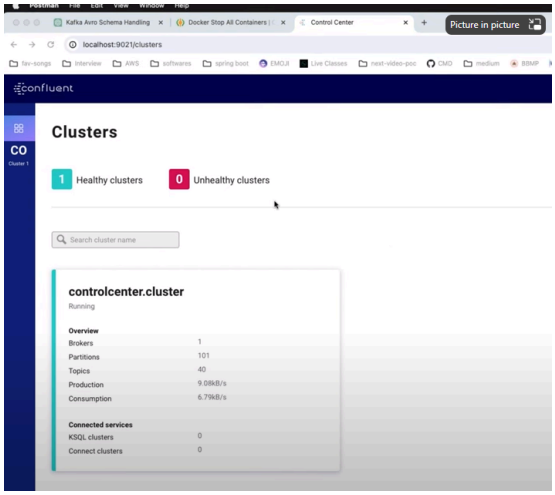
Code | Blame | 75 lines (70 loc) · 2.33 KB

```
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

  broker:
    image: confluentinc/cp-server:5.4.0
    hostname: broker
    container_name: broker
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_METRIC_REPORTERS: io.confluent.metrics.reporter.ConfluentMetricsReporter
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 1
      CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: broker:29092
      CONFLUENT_METRICS_REPORTER_ZOOKEEPER_CONNECT: zookeeper:2181
      CONFLUENT_METRICS_REPORTER_TOPIC_REPLICAS: 1
      CONFLUENT_METRICS_ENABLE: "true"
      CONFLUENT_SUPPORT_CUSTOMER_ID: "anonymous"

  kafka-tools:
    image: confluentinc/cp-kafka:5.4.0
    hostname: kafka-tools
    container_name: kafka-tools
    command: ["tail", "-f", "/dev/null"]
    network_mode: "host"

  schema-registry:
    image: confluentinc/cp-schema-registry:5.4.0
    hostname: schema-registry
    container_name: schema-registry
    depends_on:
      - zookeeper
      - broker
```

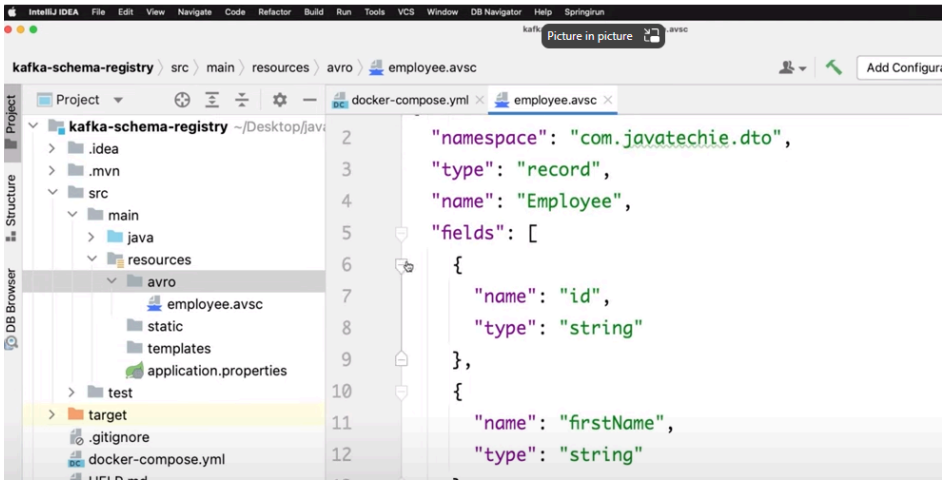
```
  kafka-tools:
    image: confluentinc/cp-kafka:5.4.0
    hostname: kafka-tools
    container_name: kafka-tools
    command: ["tail", "-f", "/dev/null"]
    network_mode: "host"

  schema-registry:
    image: confluentinc/cp-schema-registry:5.4.0
    hostname: schema-registry
    container_name: schema-registry
    depends_on:
      - zookeeper
      - broker
    ports:
      - "8081:8081"
    environment:
      SCHEMA_REGISTRY_HOST_NAME: schema-registry
      SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: "zookeeper:2181"

  control-center:
    image: confluentinc/cp-enterprise-control-center:5.4.0
    hostname: control-center
    container_name: control-center
    depends_on:
      - zookeeper
      - broker
      - schema-registry
    ports:
      - "9021:9021"
    environment:
      CONTROL_CENTER_BOOTSTRAP_SERVERS: 'broker:29092'
      CONTROL_CENTER_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      CONTROL_CENTER_SCHEMA_REGISTRY_URL: "http://schema-registry:8081"
      CONTROL_CENTER_REPLICATION_FACTOR: 1
      CONTROL_CENTER_INTERNAL_TOPICS_PARTITIONS: 1
      CONTROL_CENTER_MONITORING_INTERCEPTOR_TOPIC_PARTITIONS: 1
      CONFLUENT_METRICS_TOPIC_REPLICATION: 1
      PORT: 9021
```



Controller center ui



Step 1: Schema definition

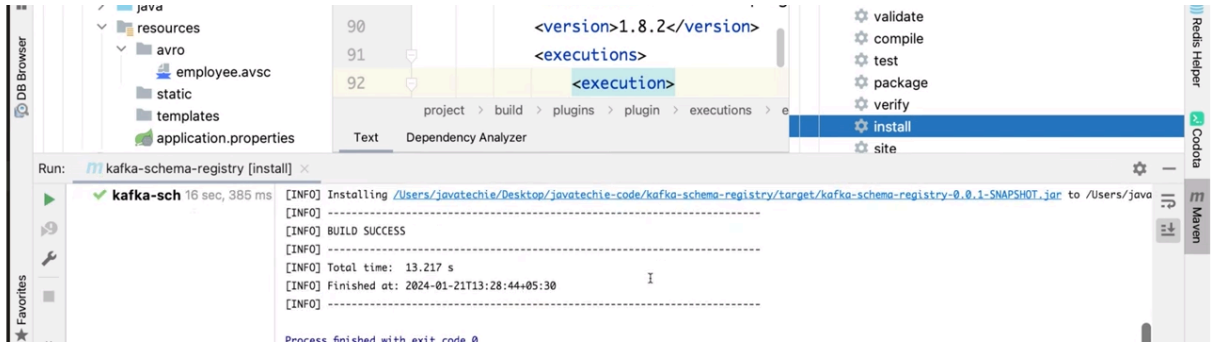


Step 2: Add Avro plugin

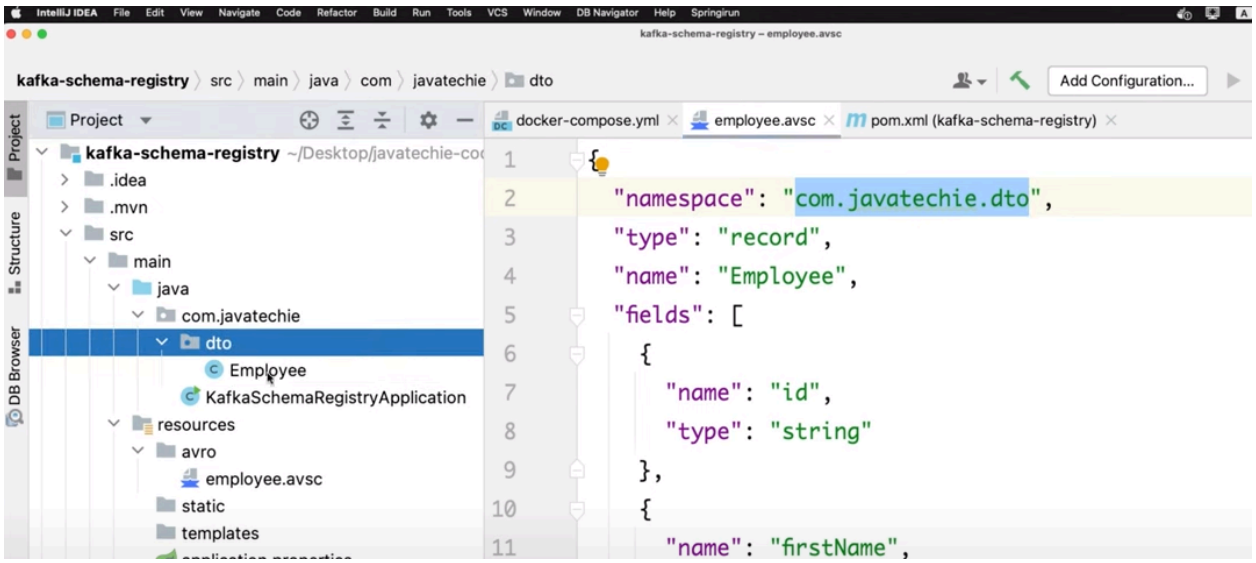
```
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>1.8.2</version>
                <executions>
                    <execution>
                        <id>schemas</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>
                            <sourceDirectory>${project.basedir}/src/main/resources/</sourceDirectory>
                            <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

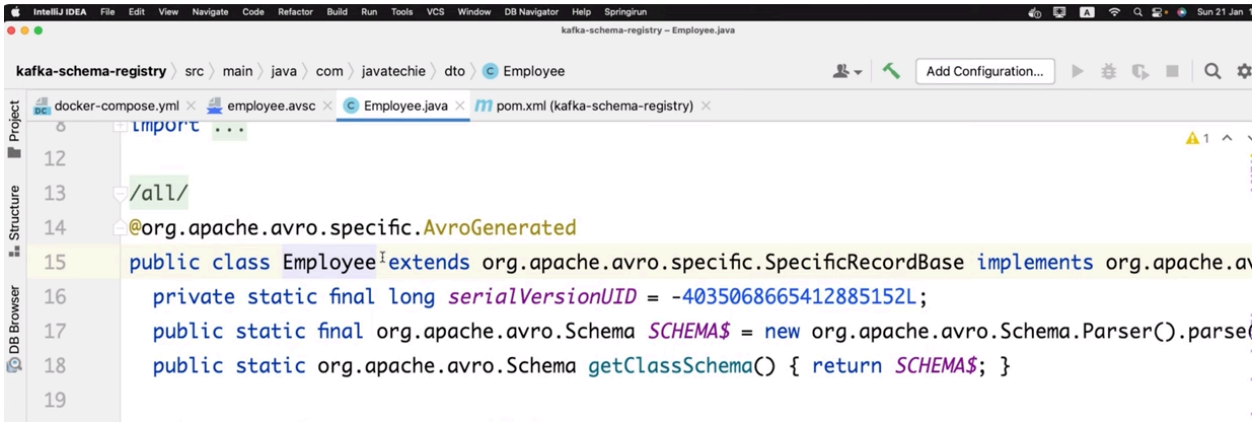
Here we defined what is the source and destination.

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

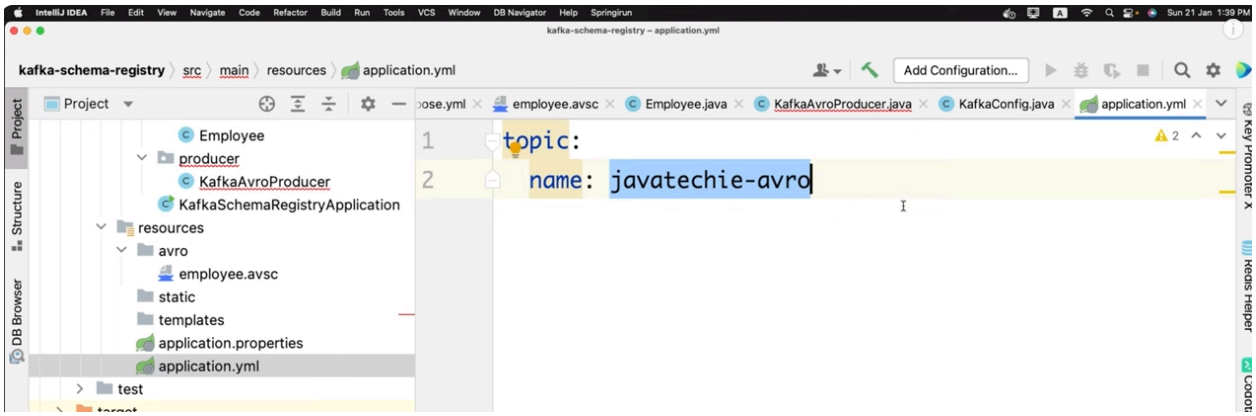


Java package and class will be created as defined in schema





Step 4: Create producer



Create a topic in configuration class as shown below.

Code Blame 18 lines (14 loc) · 462 Bytes

```java
package com.javatechie.config;

import org.apache.kafka.clients.admin.NewTopic;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class KafkaConfig {

    @Value("${topic.name}")
    private String topicName;

    @Bean
    public NewTopic createTopic(){
        return new NewTopic(topicName, 3, (short) 1);
    }
}
```

Use this topic in producer as shown below

Code Blame 35 lines (28 loc) · 1.15 KB

```java
package com.javatechie.producer;

import com.javatechie.dto.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.support.SendResult;
import org.springframework.stereotype.Service;

import java.util.UUID;
import java.util.concurrent.CompletableFuture;

@Service
public class KafkaAvroProducer {

    @Value("${topic.name}")
    private String topicName;

    @Autowired
    private KafkaTemplate<String, Employee> template;


    public void send(Employee employee){
        CompletableFuture<SendResult<String, Employee>> future = template.send(topicName, UUID.randomUUID().toString(),employee);
        future.whenComplete((result, ex) -> {
            if (ex == null) {
                System.out.println("Sent message=[" + employee +
                        "] with offset=[" + result.getRecordMetadata().offset() + "]");
            } else {
                System.out.println("Unable to send message=[" +
                        employee + "] due to : " + ex.getMessage());
            }
        });
    }
}
```

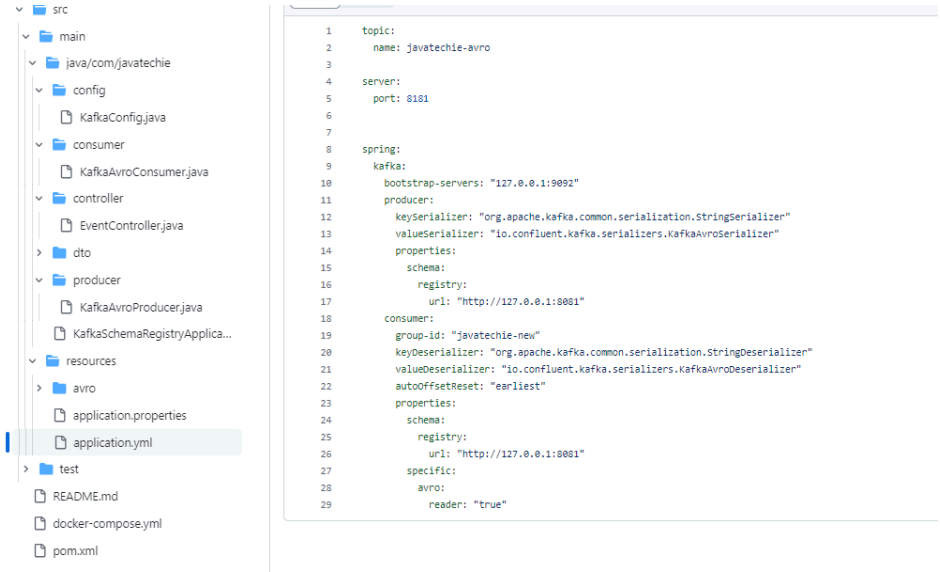
Step 5: Create consumer as shown below

```java
package com.javatechie.consumer;

import com.javatechie.dto.Employee;
import lombok.extern.slf4j.Slf4j;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
@Slf4j
public class KafkaAvroConsumer {

    @KafkaListener(topics = "${topic.name}")
    public void read(ConsumerRecord<String, Employee> consumerRecord) {
        String key = consumerRecord.key();
        Employee employee = consumerRecord.value();
        log.info("Avro message received for key : " + key + " value : " + employee.toString());

    }
}
```

Create port for this app

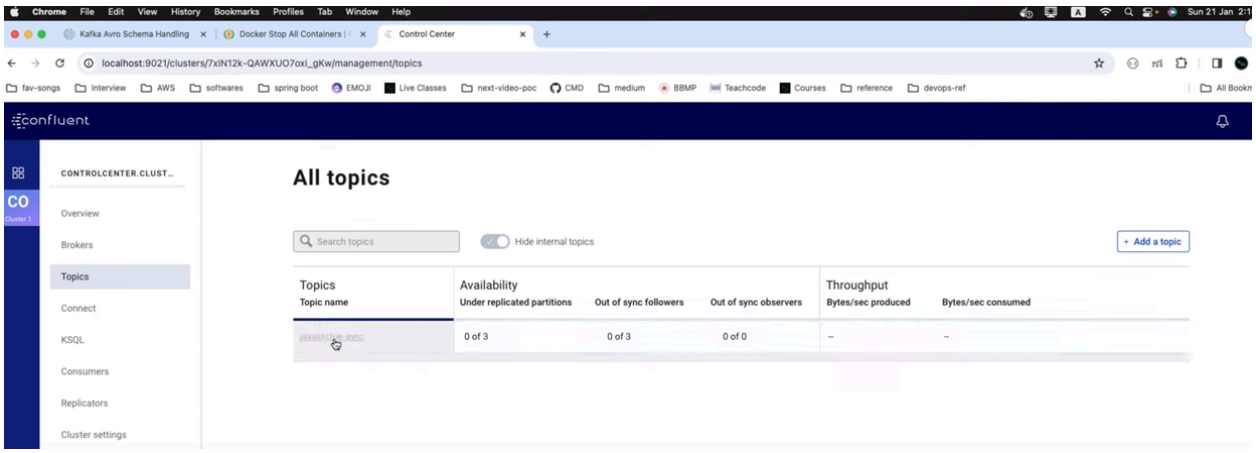
Step 6: Create producer Avro serializer and its schema registry address.

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

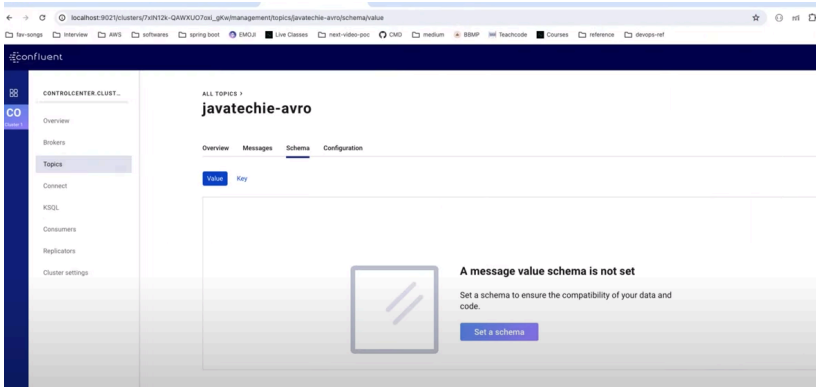


Now All the steps are completed let us start the application and test it

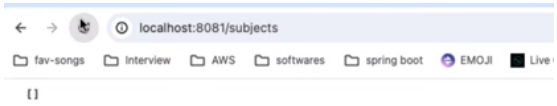
Topic with two partitions was created.



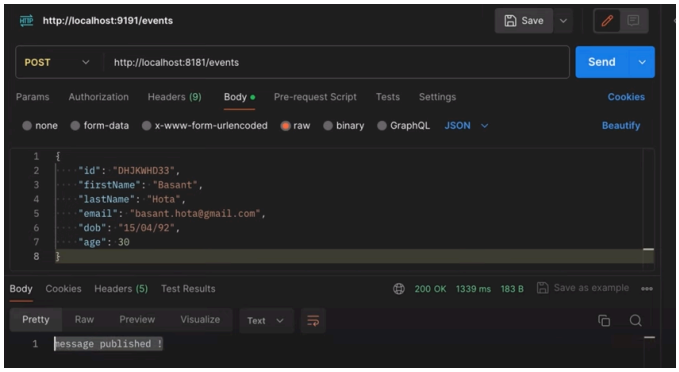
At the beginning we may not see schema in the control-center. Once we publish it will register.

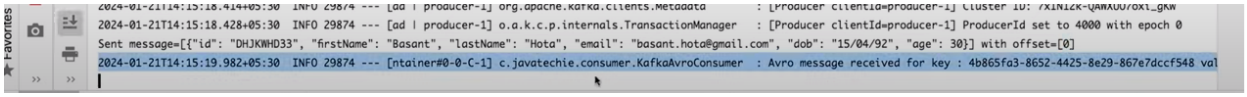


Not even in schema registry also we won't see

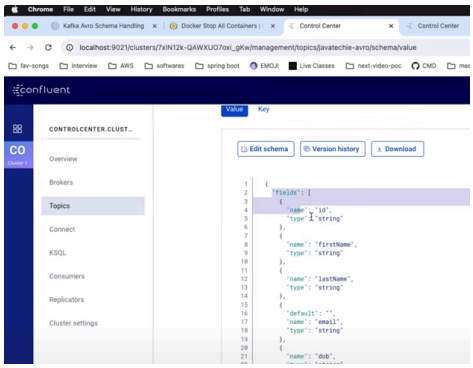


Publish a message from postman

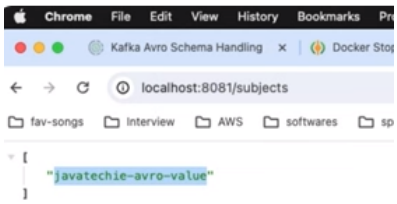




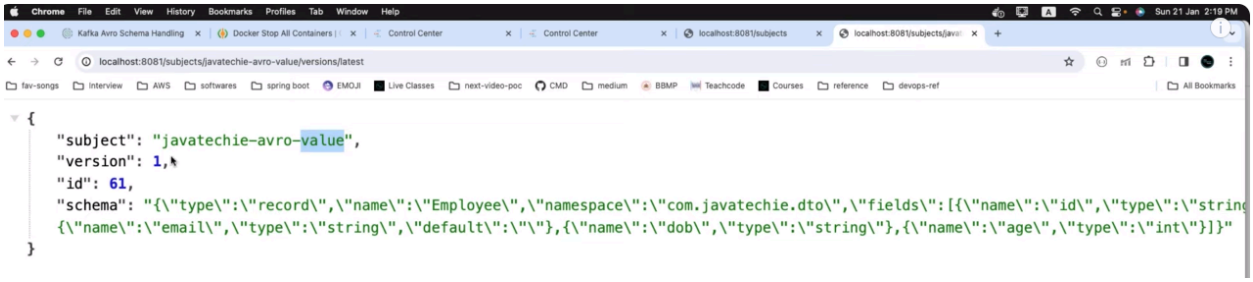
Now we can see schema



In schema registry, The below endpoint will return topic

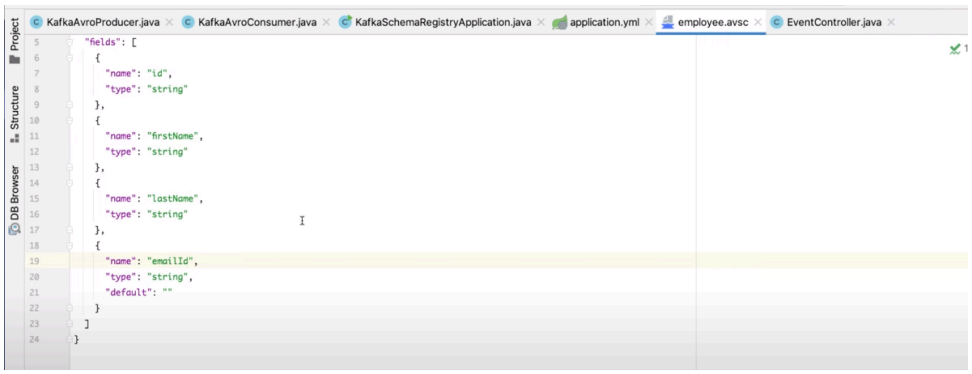


The below endpoint will return avro schema



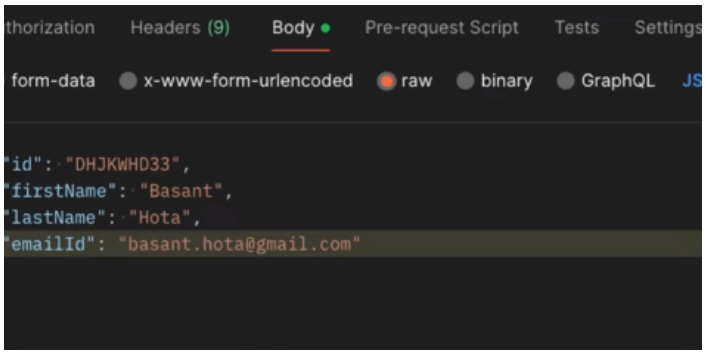
Version of the schema is 1.

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

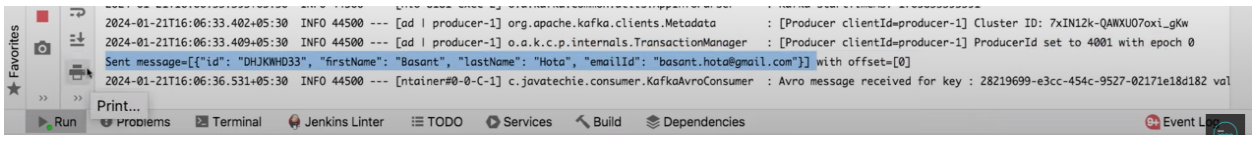


Regenerate the avro

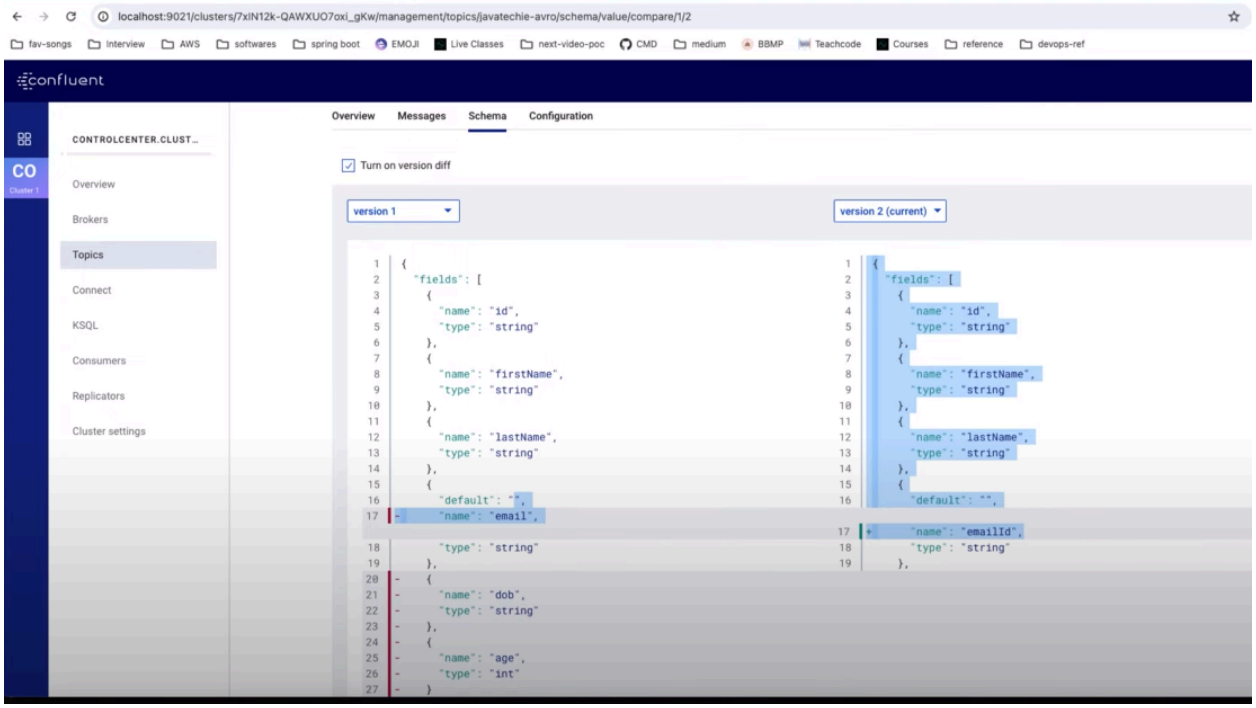
start the application



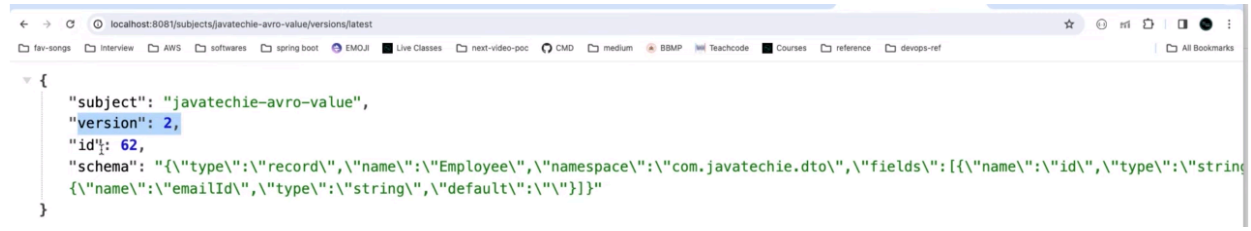
Publish the message



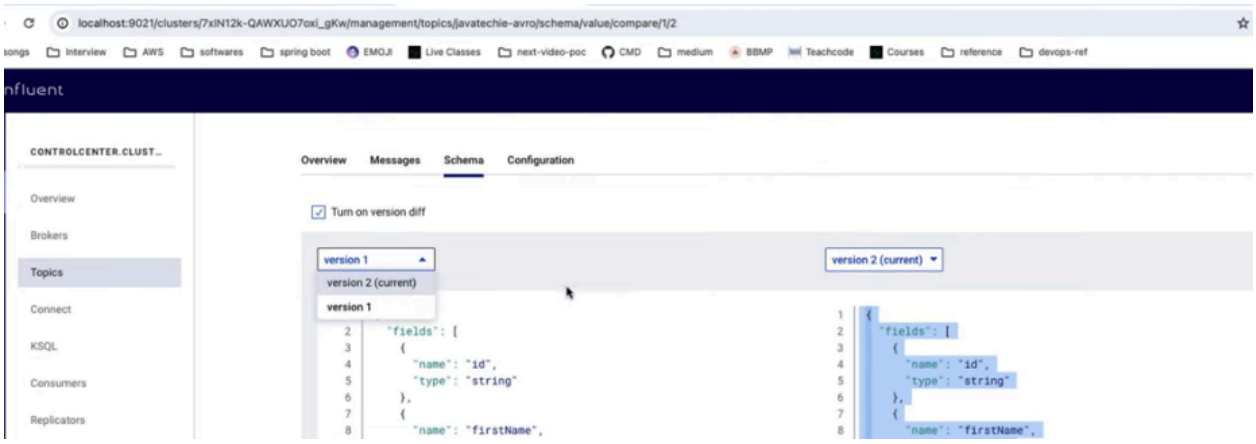
We can see the comparison between previous and current versions in the control center as shown below.



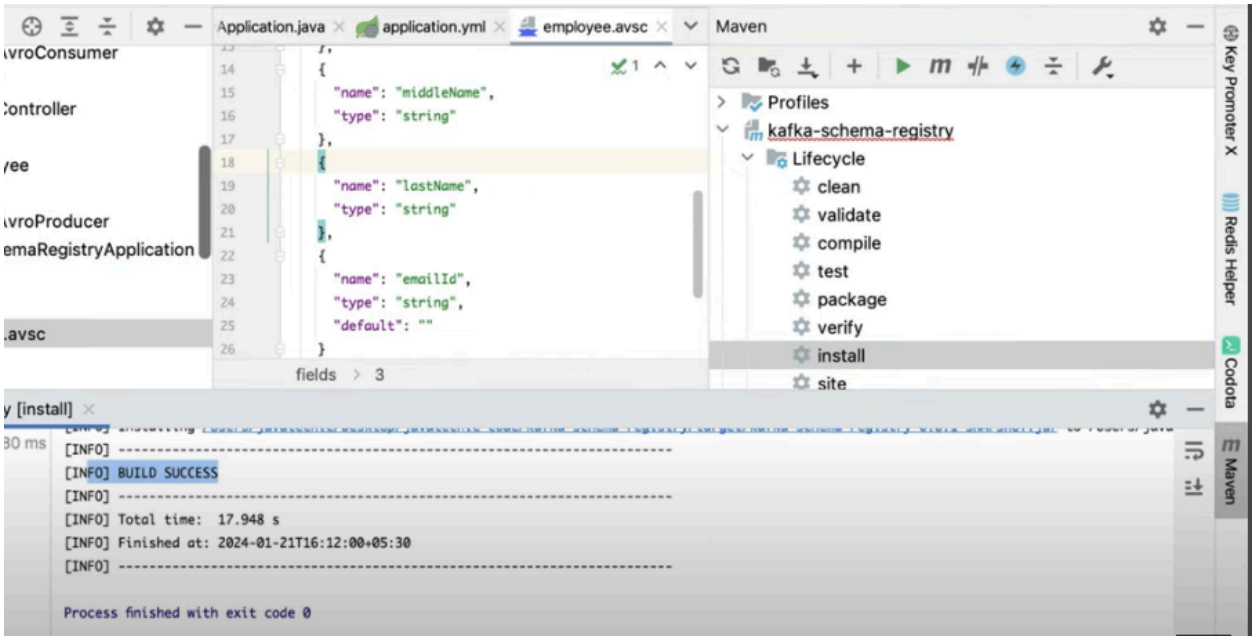
In the schema registry we see a new version.



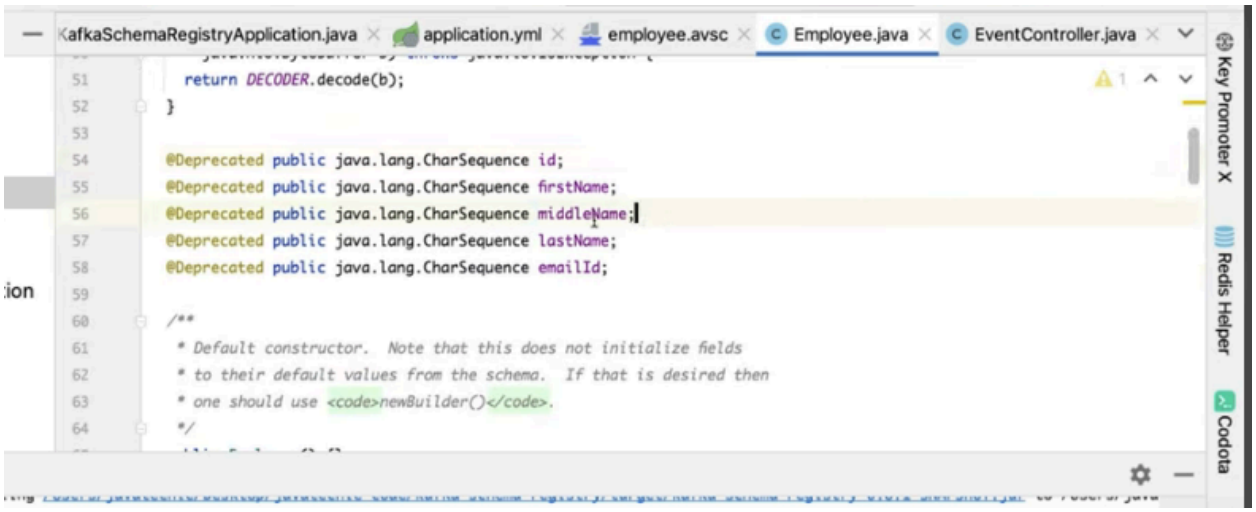
It maintain schema history



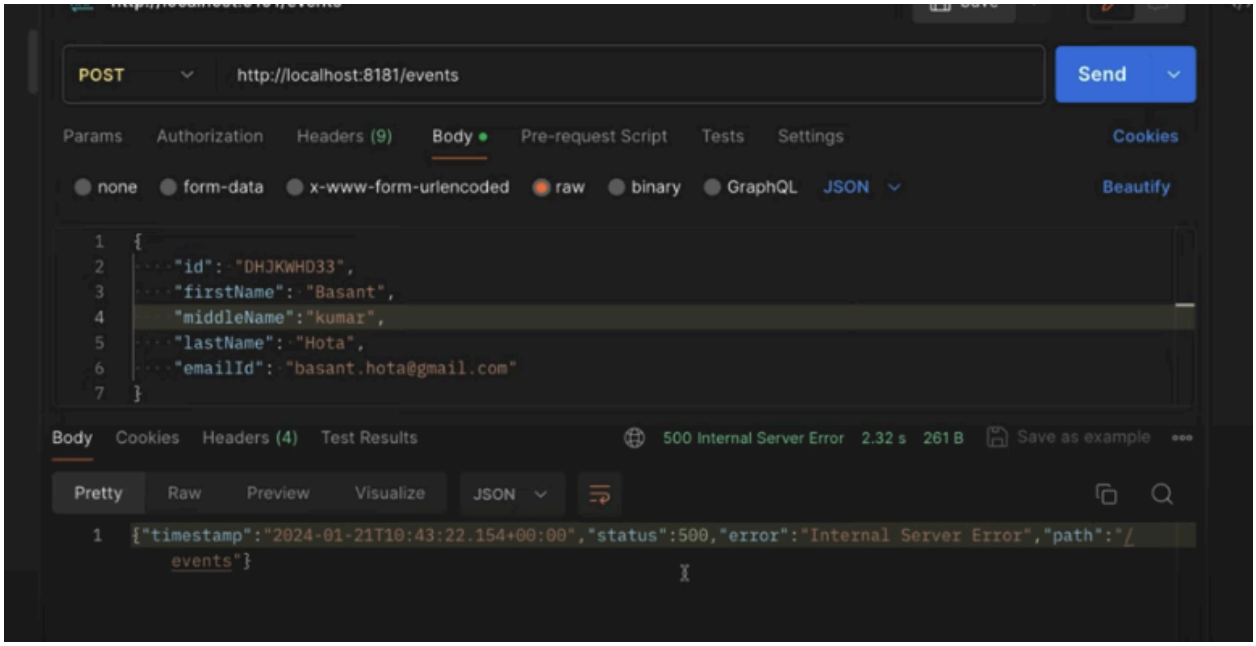
Add middle name and maven install

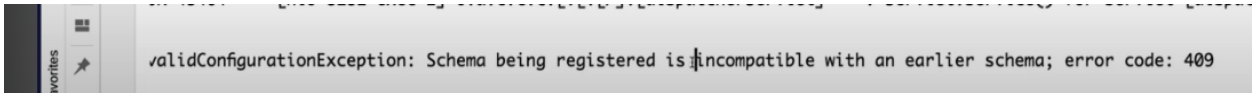


Middle name is added in java file.

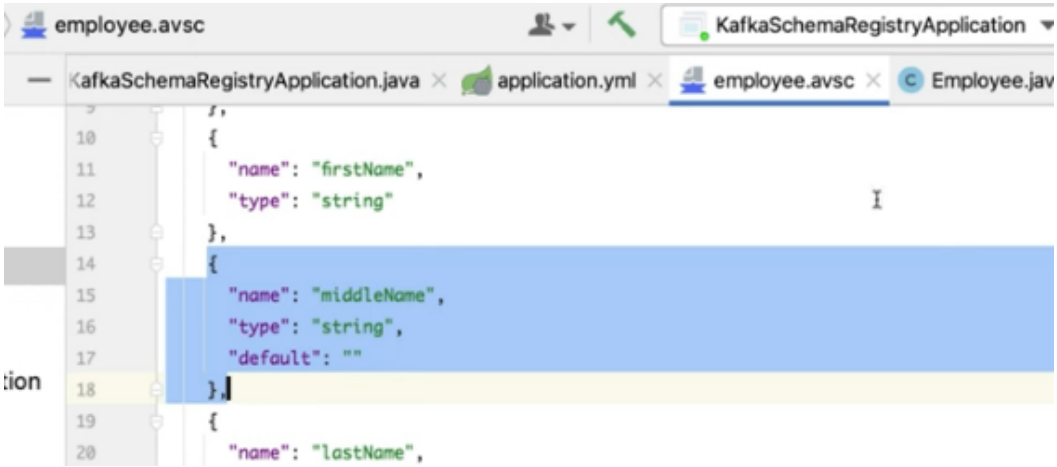


We got error with this new change

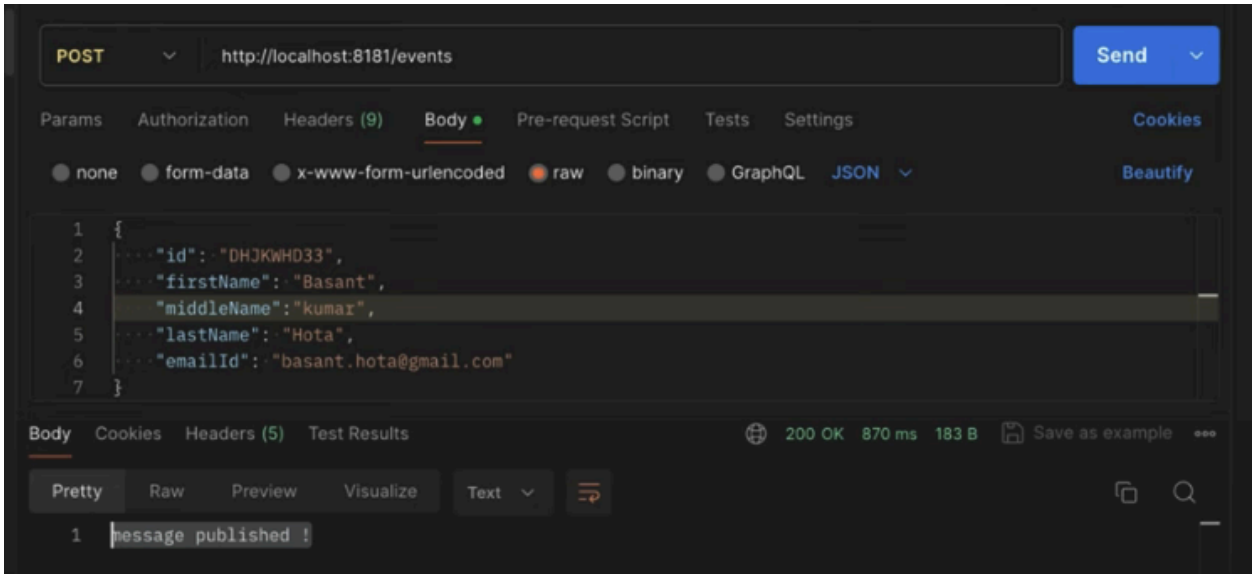


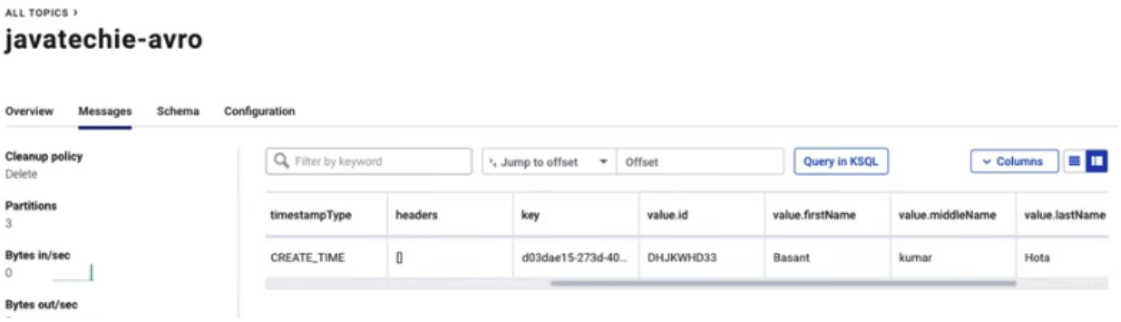


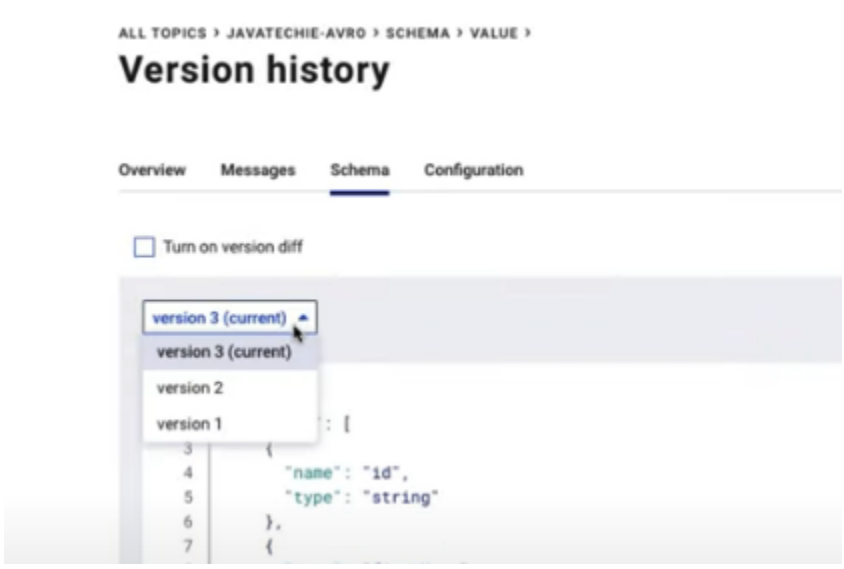
For any new field mark it as default as shown below.



Rebuild the project and run







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