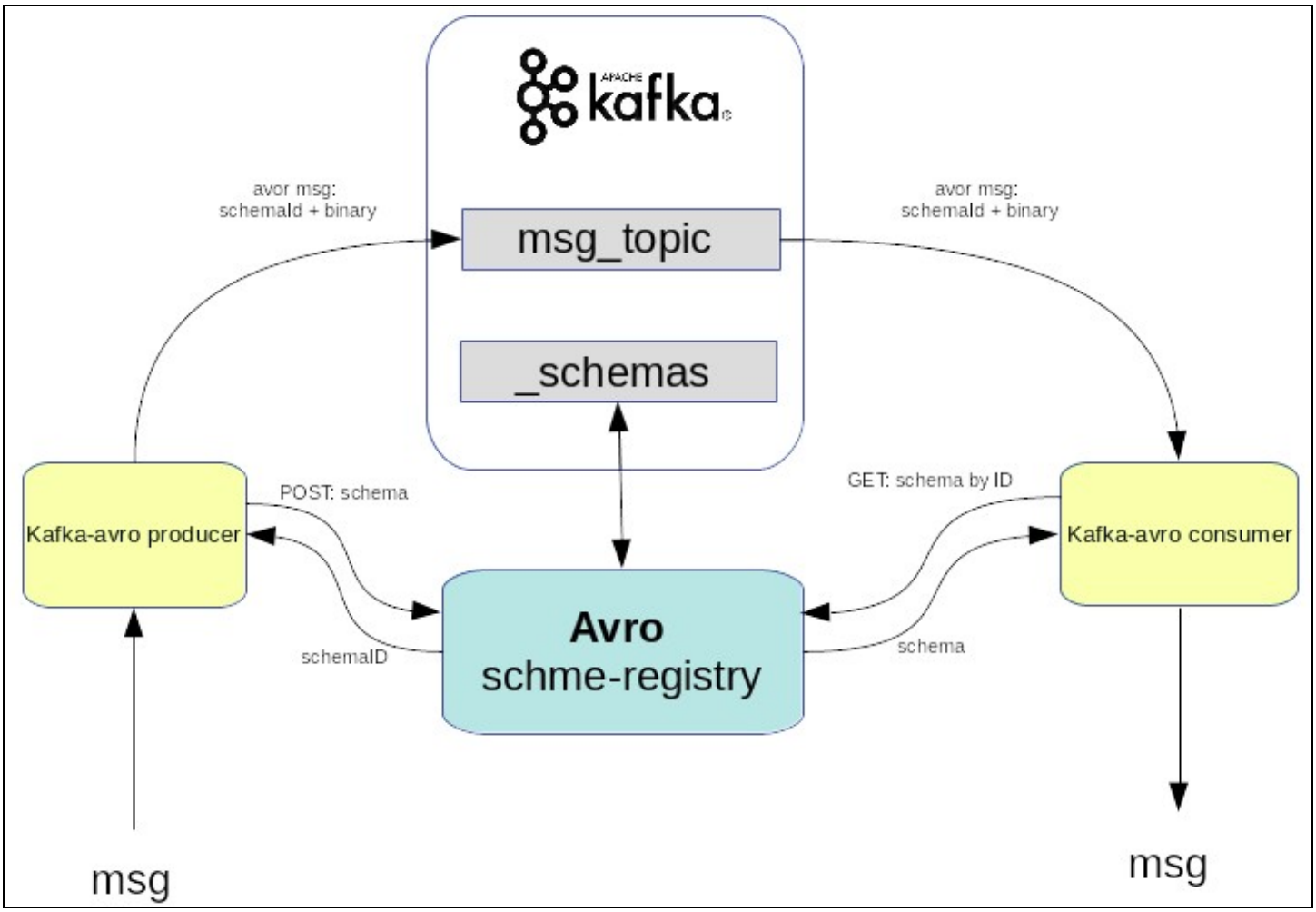
Apache Avro with Kafka

Contents

Bevezet?

Mi az Avro?

Az Avro egy nyílt forráskódú project, ami egy adata szerializációs szolgáltatás els?sorban az Apache Hadoop-hoz, de nem csak a Hadoop-ban használható, ahogy a mi példánkban is látni fogjuk. Az Avro segítségével nagyon hatékonyan cserélhetünk adatokat két végpont között "big data" környezetben.

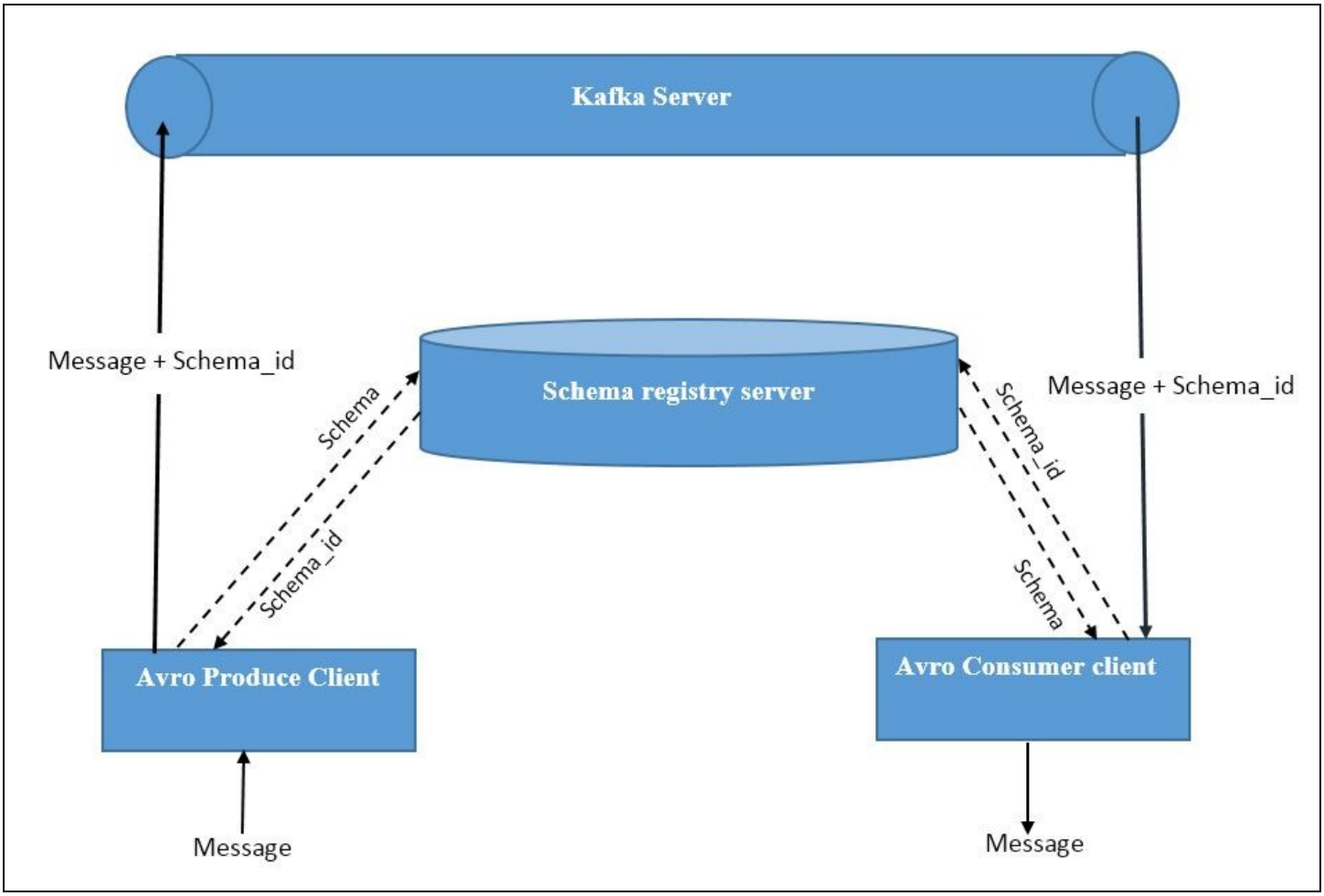


Az Avro alapja egy séma regiszter, amihez mind az adat szerializáló és deszerializáló szolgáltatás kapcsolódik. Itt tárolja az Avro a serializálnadó adatok tervrajzát JSON formátumban, a sémákat verziózva. Amikor a serializáló szolgáltatás adatot akar küldeni, akkor megjelöli az Avro serializátornak hogy melyik séma alapján serializálja a küldend? adatot. Ha a séma még nem létezett, akkor beszúrja a séma regiszterbe. Az Avro binárist készít a séma segítségével a küldend? adatból, és az üzenetbe a bináris adat mellé beleteszi a séma azonosítóját is, amit a deszerializiós szolgáltatás megkap, és annak segítségével ki tudja olvasni az adat deszerializálásához szükséges sémát, ami segítségével el?állítja az eredeti üzenetet.

Az Avro séma regiszter több verziót is képes kezelni egy sémából. A beállításoknak megfelel?en a séma lehet el?re vagy visszafelé kompatibilis. Ha egy séma visszafelé kompatibilis, akkor az új sémával is ki lehet olvasni olyan régi adatokat, amit még egy korábbi sémával írtak be.

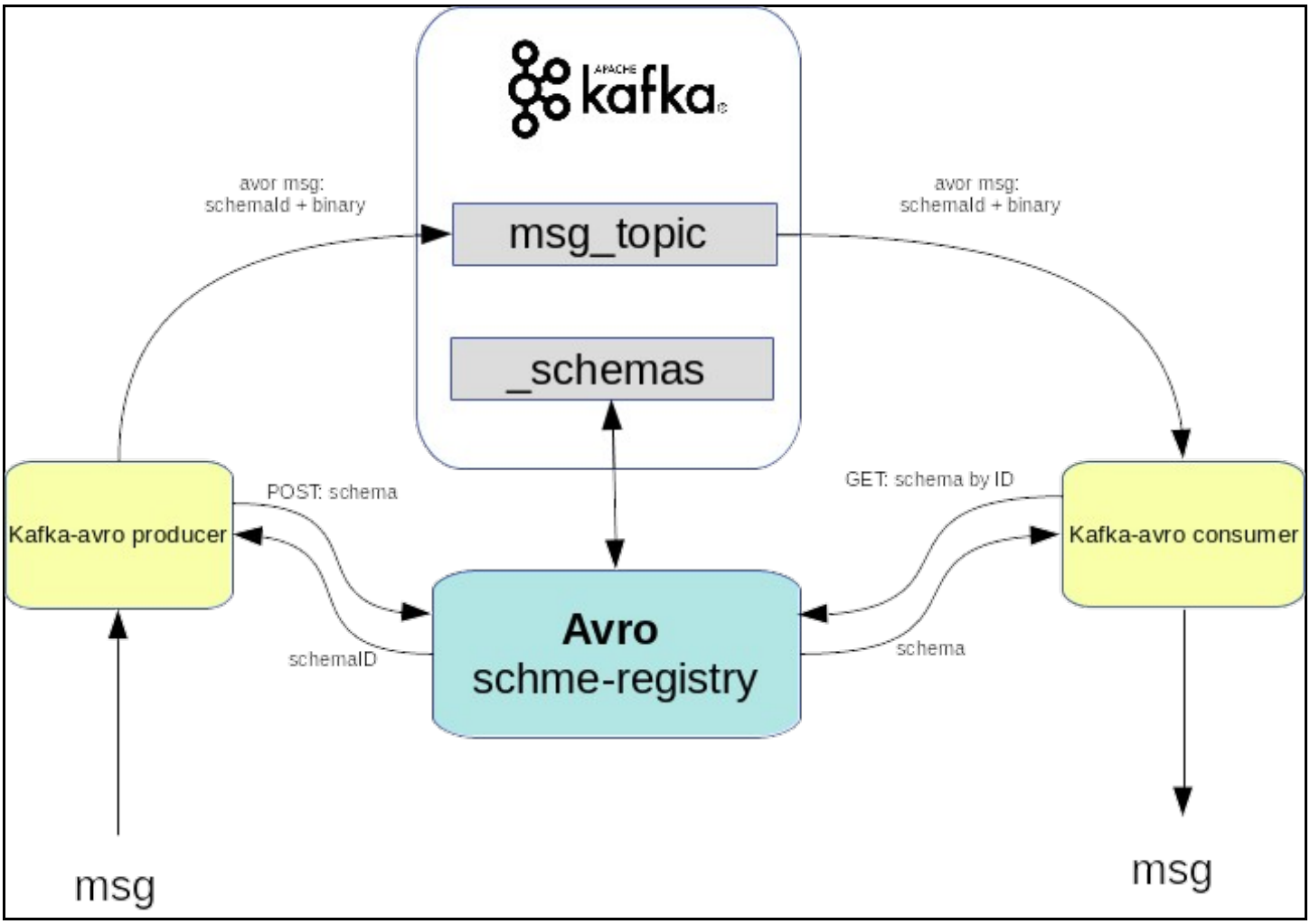
Az Avor sémákat JSON formátumban kell leírni, aminek egyedi, kötött szintaktikájuk van, tehát nem szabványos JSON sémák. A schema-registry-el egy REST API-n keresztül lehet kommunikálni. A legtöbb programozási nyelven elérhet? Avro magas szint? API, ami elfedi el?lünk a REST kommunikációt. Az Avró többféle adatbázisban is tárolhatja a sémákat, de a leggyakoribb megoldás, hogy egy speciális Kafka topic-ban tárolja azokat.

Az Avro-t gyakran használják a Kafa kommunikációban mint serlializációs szólgáltatás. Mi is így fogjuk használni:



A Kafa produceren a Avro serializáló beküldi a sémát a séma regiszterbe, aminek visszakapja az ID-ját. Majd a séma alapján serializálja az adatokat és gyárt bel?le egy Avro üzenetet, amiben benne van a séma azonosító és binárisan az üzenet, ami így nagyon kicsi helyet foglal. Ez kerül fel a megfelel? Kafka topic-ra. A Kafka consumer az üzenetben lév? séma ID alapján lekérdezi a sémát a registry-b?l, majd annak segítségével deserializálja az üzenetet.

Ahogy azt már írtam, az Avro schema-registry többféle adatbázisban is képes tárolni a sémákat, köztük Kafka-ban is. Kafka estén az Avro regiszter egy kitüntetett topic-ban tárolja a sémákat (_schemas). Ha a kommunikációra is Kafka-t használunk, akkor használhatjuk akár ugyan azt a Kafka broker-t mind két célra, ahogy az alábbi ábrán is látható:



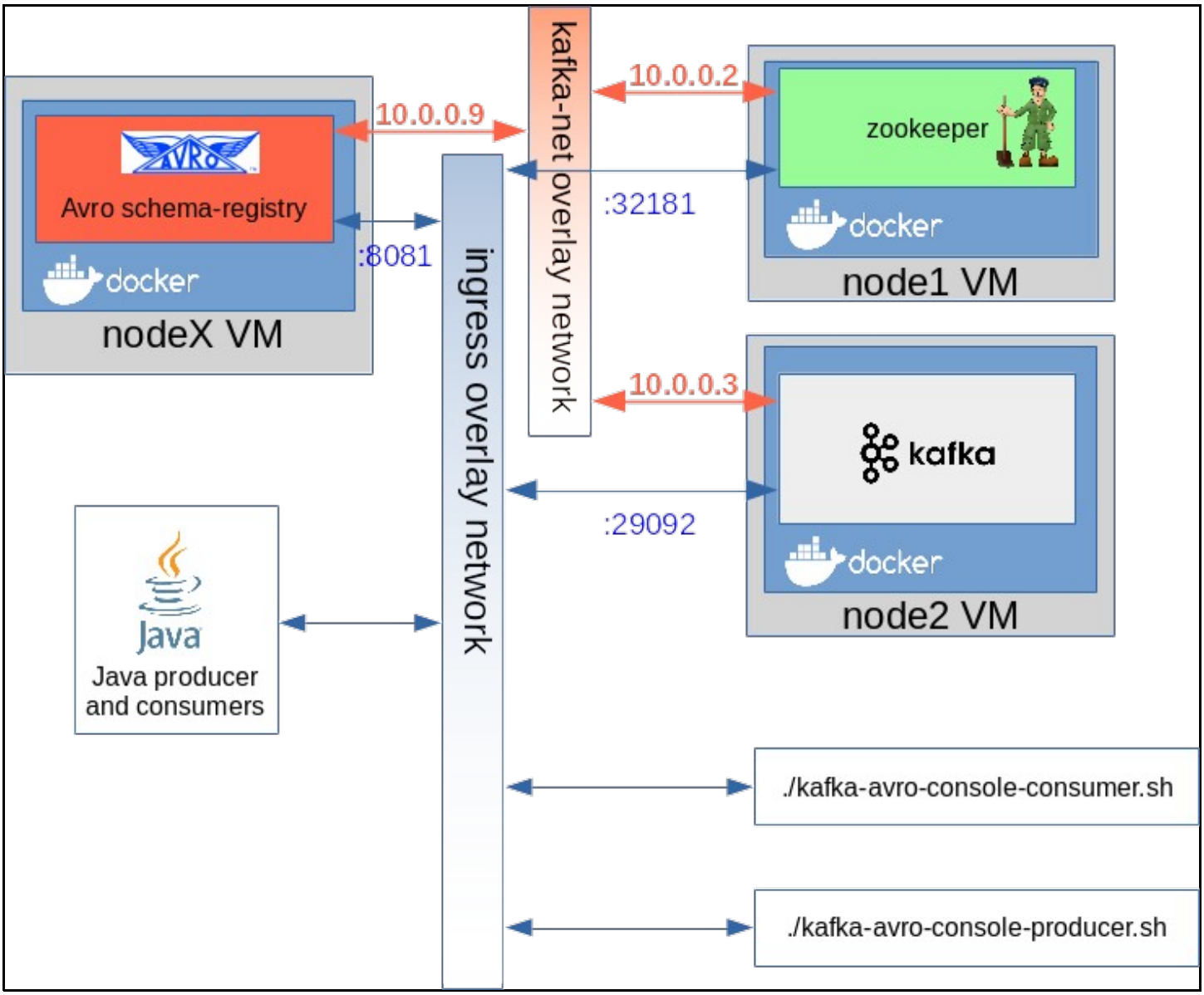
Környezet

Az Avro futtatásához szükséges környezet egy két node-os swarm cluster lesz.

```
# virsh list
 Id    Name                           State
----------------------------------------------------
 1     mg0                            running
 2     worker0                        running

# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
maigxlyagj1fl4sgcf6rnn9pc *   mg0                 Ready               Active              Leader              18.05.0-ce
vox99u5s1g1su742mc6npm370     worker0             Ready               Active                                  18.05.0-ce
```

Itt fogunk futtatni egy docker stack-et ami tartalmaz majd egy avro schema-registry-t, egy kafa brókert és egy zookeeper példányt.



confluent_swarm.yaml

```
version: '3.2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.1.2
    networks:
      - kafka-net
    ports:
      - "32181:32181"
    deploy:
      placement:
        constraints:
         - node.role == worker
    environment:
      ZOOKEEPER_CLIENT_PORT: 32181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_SYNC_LIMIT: 2
  kafka:
    image: confluentinc/cp-kafka:5.1.2
    networks:
      - kafka-net
    ports:
      - target: 29092
        published: 29092
        protocol: tcp
    deploy:
      placement:
        constraints:
         - node.role == worker
    environment:
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:32181"
      KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka:29092"
      KAFKA_BROKER_ID: 2
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  schemaregistry:
    image: confluentinc/cp-schema-registry:5.1.2
    networks:
      - kafka-net
```

```
    ports:
      - target: 8081
        published: 8081
        protocol: tcp
    deploy:
      placement:
        constraints:
          - node.role == worker
    environment:
      SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: "zookeeper:32181"
      SCHEMA_REGISTRY_HOST_NAME: "schemaregistry"
      SCHEMA_REGISTRY_DEBUG: "true"
networks:
  kafka-net:
    driver: overlay
```

Hozzuk létre a docker stack-et:

```
# docker stack deploy -c confluent_swarm.yaml confluent
```

Listázzuk ki a stack-ban létrejött service-eket és az overlay hálózatot:

```
# docker service ls
ID                  NAME                       MODE                REPLICAS            IMAGE                                        PORTS
7vjvop7tqiyc        confluent_kafka            replicated          1/1                 confluentinc/cp-kafka:5.1.2                  *:29092->29092
in6a4ti3jeu5        confluent_schemaregistry   replicated          1/1                 confluentinc/cp-schema-registry:5.1.2        *:8081->8081/t
oxxjtkcusj1f        confluent_zookeeper        replicated          1/1                 confluentinc/cp-zookeeper:5.1.2              *:32181->32181

# docker network ls
NETWORK ID          NAME                   DRIVER              SCOPE
...
5albky0eu1to        confluent_kafka-net    overlay             swarm
olqkh5zlqiac        ingress                overlay             swarm
...
```

Keressük meg a worker0 node IP címét:

```
# docker-machine ip worker0
192.168.42.42
```

Mivel mind a három komonensünknek egy-egy portját publikáltuk az ingress loadbalance-olt hálózatra, ezrét az összes node-on elérhet?ek az adott portokon.

- zookeeper: 192.168.42.42:32181
- kafka: 192.168.42.42:29092
- schema-registry: 192.168.42.42:8081

Avro REST interfész

Az Avro a **_schemas** nev? Kafka topic-ban tárolja a sémákat az alapértelmezett konfiguráció szerint. A Kafka /bin mappájában található **kafka-topics.sh** topic admin script-el listázzuk ki a topikokat:

```
$ ./kafka-topics.sh --list --zookeeper 192.168.42.42:32181
__confluent.support.metrics
_schemas
```

Az schma registry-vel a REST interfészén keresztül lehet kommunikálni. Próbáljuk ki a config paranccsal hogy elérhet? e a server a host gépr?l.

```
$ curl -X GET http://192.168.42.42:8081/config
{"compatibilityLevel":"BACKWARD"}
```

A válaszban láthatjuk a kompatibilitási szintet (ezt majd kés?bb részletesen tárgyaljuk).

Avro-ban minden sémát egy úgynevezett subject-ek alá kell regisztrálni. Egy subject alatt ugyan azon séma különböz? verzióit tároljuk. Tehát két teljesen különböz? sémát nem lehet ugyanazon subject alá berakni. Tehát mikor hasonló sémákat regisztrálunk ugyanazon subject alá, akkor különböz? verziók fognak létrejönni ugyan ahhoz a sémához. Azt hogy mekkora a megengedett eltérés mértéke, a schema-registry server konfigurációja határozza meg.

The schema registry server can enforce certain compatibility rules when new schemas are registered in a subject. Currently, we support the following compatibility rules.

- **Backward compatibility** (default): A new schema is backward compatible if it can be used to read the data written in all previous schemas. Backward compatibility is useful for loading data into systems like Hadoop since one can always query data of all versions using the latest schema.
- **Forward compatibility**: A new schema is forward compatible if all previous schemas can read data written in this schema. Forward compatibility is useful for consumer applications that can only deal with data in a particular version that may not always be the latest version.
- **Full compatibility**: A new schema is fully compatible if it?s both backward and forward compatible.
- **No compatibility**: A new schema can be any schema as long as it?s a valid Avro.

A sémákat a POST:/subjects/<subject-name>/versions REST interfészen kell beküldeni. A POST body-ban a {schema: "...séma definició..."} formátumban kell megadni a sémát, ahol a séma definíció egy escape-lt bels? json.

```
$ curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema" : "... schema def..."}' http://192.168.42.42:8081/
```

Szúrjuk be az Avro-ba az alábbi **Employee** sémát. A namespace majd a schema-to-java kód generálásánál lesz érdekes, ez fogja meghatározni a java csomagot generált kódban. A type mez? mutatja meg, hogy összetett objektumot, sima stringet, vagy tömböt ír le a séma. A **record** jelenti az összetett objektumot. Az **Employee** nev? objektum négy mez?b?l áll.

```
{"namespace": "hu.alerant.kafka.avro.message",
  "type": "record",  "name": "Employee",
    "fields": [
        {"name": "firstName", "type": "string"},
        {"name": "lastName", "type": "string"},
        {"name": "age",  "type": "int"},
        {"name": "phoneNumber",  "type": "string"}
    ]
}
```

Ennek az escape-elt változata:

```
{\"namespace\": \"hu.alerant.kafka.avro.message\",  \"type\": \"record\",  \"name\": \"Employee\",   \"fields\": [ {\"name\": \"firstName\",
```

Szurjuk be a fenit sémát a **test1** subject alá:

```
$ curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema" : "{\"namespace\": \"hu.alerant.kafka.avro.message

{"id":1}
```

A válaszban visszakaptuk a séma példány egyedi azonosítóját. Ez nem összekeverend? a séma verziójával. Tehát a **test1** subject-en ugyan annak a sémának több verziója is lehet, de globálsan, ennek s séma példánynak az ozonosítója = 1

Most próbáljunk az el?bbit?l tejesen különböz? **Company** sémát regisztrálni szintán a **test1** subject alá.

```
{"namespace": "hu.alerant.kafka.avro.message",
  "type": "record",  "name": "Company",
    "fields": [
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"},
        {"name": "employCount",  "type": "int"},
        {"name": "phoneNumber",  "type": "string"}
    ]
}
```

Ennek az escape-elt változata az alábbi.

```
{\"namespace\": \"hu.alerant.kafka.avro.message\",  \"type\": \"record\",  \"name\": \"Company\",   \"fields\": [ {\"name\": \"name\", \"type
```

A **Company** sémát szúrjuk be szintén az **test1** subject alá.

```
$ curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema" : "{\"namespace\": \"hu.alerant.kafka.avro.message

{"error_code":409,"message":"Schema being registered is incompatible with an earlier schema ...}
```

Láthatjuk, hogy nem engedte az Avro a **Company** sémát regisztrálni a **test1** subject alá, mert túl nagy volt az eltérés a **Company** és a **Employee** sémák között.

Láthattuk a /config lekérdezésben, hogy jelenleg a beállított kompatibilitási szint **BACKWARD**, ami azt jelenti, hogy csak olyan sémákat lehet beszúrni ugyan azon subject alá, amivel az összes korábban beszúrt adatot ki lehet olvasni, magyarán csak olyan sémákat lehet egymás után beszúrni, ami részhalmaza az el?z? sémának.

Most szúrjuk be az **Employee** sémának egy redukált változatát, amib?l hiányzik a **phoneNumber** mez?. Erre teljes?l hogy visszafelé komatibilis.

```
{"namespace": "hu.alerant.kafka.avro.message",
  "type": "record",  "name": "Employee",
    "fields": [
        {"name": "firstName", "type": "string"},
        {"name": "lastName", "type": "string"},
        {"name": "age",  "type": "int"}
    ]
}
```

Ennek az escape-elt változata az alábbi:

```
{\"namespace\": \"hu.alerant.kafka.avro.message\",\"type\": \"record\",  \"name\": \"Employee\", \"fields\": [ {\"name\": \"firstName\", \"ty
```

Szúrjuk ezt be a **test1** subject alá:

```
$ curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" --data '{"schema" : "{\"namespace\": \"hu.alerant.kafka.avro.message

{"id":2}
```

Láthatjuk, hogy az új séma példány egyedi azonosítója 2.

Most listázzuk ki a **test1** subject-en belül az összes sémát:

```
$ curl -X GET -H "Content-Type: application/vnd.schemaregistry.v1+json" http://192.168.42.42:8081/subjects/test1/versions

[1,2]
```

Láthatjuk, hogy két verziója van elmentve a sémának, amiknek a globális azonosítója 1 és 2.

Ha a /versions/ után odaírjuk a verzió számot is, akkor visszaadja a teljes sémát:

```
$ curl -X GET -H "Content-Type: application/vnd.schemaregistry.v1+json" http://192.168.42.42:8081/subjects/test1/versions/1

{"subject":"test1","version":1,"id":1,"schema":"{\"type\":\"record\",\"name\":\"Employee\",\"namespace\":\"hu.alerant.kafka.avro.message\",\"
```

Java kód generálás

Van egy maven plugin, amivel a sémából ki lehet genrálni az Avro-s java osztályokat, amiket majd használni tudunk mind a java producer és consumer-ben. A fenit .xml sémákat tegyük be a /schemas/ mappába .avsc kiterjesztésben:

- Employee.avsc
- Company.avsc

A forrást a /src/main/java/ mappába fogja tenni. Az avro által generált java osztály package a sémában lév? **namespace** értéke lesz.

pom.xml

```
<dependency>
<groupId>org.apache.avro</groupId>
<artifactId>avro-maven-plugin</artifactId>
<version>1.8.2</version>
</dependency>
<dependency>
<groupId>org.apache.avro</groupId>
<artifactId>avro-compiler</artifactId>
<version>1.8.2</version>
</dependency>
<dependency>
<groupId>org.apache.avro</groupId>
<artifactId>avro-ipc</artifactId>
<version>1.8.2</version>
</dependency>
```

```
<dependency>
<groupId>org.apache.avro</groupId>
<artifactId>avro</artifactId>
<version>1.8.2</version>
</dependency>
....
<build>
<plugins>
<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>
<goal>protocol</goal>
<goal>idl-protocol</goal>
</goals>
<configuration>
<sourceDirectory>${project.basedir}/schemas/</sourceDirectory>
<outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Buildelés:

```
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------------< kafka:avro >----------------------------
[INFO] Building avro 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- avro-maven-plugin:1.8.2:schema (schemas) @ avro ---
[INFO]
[INFO] --- avro-maven-plugin:1.8.2:protocol (schemas) @ avro ---
```

A generált osztályba az Avro belegenrálja a sémát is, ez az amit majd a Kafak topic-ba dobás el?tt a producer fel fog küldeni a schema-register szervernek.

Employee.java

```
package hu.alerant.kafka.avro.message;

import org.apache.avro.specific.SpecificData;
import org.apache.avro.message.BinaryMessageEncoder;
import org.apache.avro.message.BinaryMessageDecoder;
import org.apache.avro.message.SchemaStore;


@org.apache.avro.specific.AvroGenerated
public class Employee extends org.apache.avro.specific.SpecificRecordBase implements org.apache.avro.specific.SpecificRecord {

  public static final org.apache.avro.Schema SCHEMA$ = new org.apache.avro.Schema.Parser().parse("
         {\"type\":\"record\",\"name\":\"Employee\",\"namespace\":\"hu.alerant.kafka.avro.message\",
          \"fields\":[{\"name\":\"firstName\",\"type\":\"string\"},{\"name\":\"lastName\",\"type\":\"string\"},
         {\"name\":\"age\",\"type\":\"int\"},{\"name\":\"phoneNumber\",\"type\":\"string\"}]}");

....
```

Producers

Java avro-kafak producer

A hagyományos Kafka java producer-hez képest csak pár különbség van a java producer inicializálásban. Egyrészt meg kell adni, hogy mind a kulcsot, mind az üzenetet Avro-val akarjuk serializálni, másrészt meg kell adni az Avro schema-registry URL-jét. A /etc/hosts fájlba felvettük a worker0 swarm node IP címével a **schema-registry** host nevet.

```
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, io.confluent.kafka.serializers.KafkaAvroSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, io.confluent.kafka.serializers.KafkaAvroSerializer.class);
props.put(KafkaAvroSerializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "http://schema-registry:8081");
```

Els? alkalommal, mikor a producer be akar dobni egy üzenetet a Kafka topic-ba, felküldi a sémát a már látott POST:http://192.168.42.42:8081/subjects/<subject-name>/versions REST hívással, amit az avro java objektumból nyer ki. Ha a séma egy futás alatt nem változik, akkor többször nem küldi fel a sémát a schema-registry-be. A producer az avro subject nevét automatikusan képezi a topoci nevéb?l. Tehát egy topoc-ba csak a kompatibilitási szabályoknak megfelel? sémáknak megfelel? üzeneteket lehet berakni. Arra nincs mód, hogy bárhogyan is megadjuk, hogy az adott objektum melyik subject melyik verziójának kell hogy megfeleljen, ezt teljesen elfedi el?lünk az API.

Összefoglalva, egy adott Kafka topic-ba, amit kommunikációra használnunk (tehát nem a séma tárolására) csak Avro kompatibilis sémáknak megfelel? objektumokat lehet beküldeni. Nem azért mert a topic nem bírna el másik sémából gyártott bináris üzenetet, hanem azért, mert az Avro API a topic nevéb?l képzi a subject nevét, és egy subject-en belül csak kompatibilis sémákat lehet tárolni.

```
package hu.alerant.kafka.avro;

import java.util.Properties;

import org.apache.kafka.clients.producer.KafkaProducer;

import io.confluent.kafka.serializers.KafkaAvroSerializerConfig;

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.serialization.LongSerializer;

import hu.alerant.kafka.avro.message.Employee;
import io.confluent.kafka.serializers.KafkaAvroSerializer;
import java.util.Properties;
import java.util.stream.IntStream;

public class AvroProducer {

private static Producer<Long, Employee> createProducer() {
Properties props = new Properties();
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, io.confluent.kafka.serializers.KafkaAvroSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
io.confluent.kafka.serializers.KafkaAvroSerializer.class);
props.put(KafkaAvroSerializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "http://schema-registry:8081");

props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka:29092");
props.put(ProducerConfig.CLIENT_ID_CONFIG, "AvroProducer");

return new KafkaProducer<Long, Employee>(props);

}

 private final static String TOPIC = "test-topic";
    public static void main(String... args) {

        Producer<Long, Employee> producer = createProducer();
        Employee bob = Employee.newBuilder().setAge(35)
                .setFirstName("Bob")
                .setLastName("Jones")
                .setPhoneNumber("")
                .build();

        producer.send(new ProducerRecord<>(TOPIC, new Long("123456778"), bob));
        producer.flush();
        producer.close();
    }
}
```

A parancssori kafka-avro consumer segítségével fogjuk kiolvasni a java producer által küldött üzeneteket. Futtassuk le a java producer-t majd indítsuk el a parancssori consumer-t. Az avro consumer csak annyiban különbözök a sima parancssori consumer-t?l, hogy a séma regiszter címét is meg kell adni.

```
./kafka-avro-console-consumer --topic test-topic --zookeeper 192.168.42.42:32181 --property schema.registry.url="http://schema-registry:8081"
SLF4J: Class path contains multiple SLF4J bindings.
...


{"firstName":"Bob","lastName":"Jones","age":35,"phoneNumber":""}
```

Mikor Java-ból küldünk Avron-n keresztül Kafka üzeneteket, akkor a producer létre fog hozni a topic nevével prefixe-lt subjet-eket, egyet a Kafak kulcsnak és egyet a hozzá tartozó értéknek automatikusan, az els? üzenet váltás után. A fenti példa futtatása után listázzuk ki az összes Avro-s subject-et:

```
$ curl -X GET -H "Content-Type: application/vnd.schemaregistry.v1+json" http://192.168.42.42:8081/subjects/

["test-topic-value","test-topic-key","test1"]
```

Láthatjuk, hogy létrehozott a **test-topic** prefixel egy subject-et a valu-nak és a Kafka kulcsnak is.

A log-ban láthatjuk, hogy két POST kéréssel a kliens beküldte a schema-registry-nek a kulcs és a value sémáját:

```
2019-03-26 17:38:50 DEBUG RestService:118 - Sending POST with input {"schema":"\"long\""} to http://schema-registry:8081/subjects/test-topic-
2019-03-26 17:38:50 DEBUG RestService:118 - Sending POST with input {"schema":"{\"type\":\"record\",\"name\":\"Employee\",\"namespace\":\"hu.
```

Partition keys

A partíciós kulcsot nem muszáj Avro sémával megadni, ha nem összetett objektum, használhatjuk a beépített serializálókat, deserealizálókat. Láthattuk is, hogy a kulcs sémája egy darab Long típust tartalmazott.

```
{"schema":"\"long\""}
```

A fenti példában a kulcs értéke Long, ezért használhatjuk egyszer?en a **LongSerializer** osztályt.

```
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, org.apache.kafka.common.serialization.LongSerializer.class);
```

A fenti példa futtatásakor már csak a value sémáját fogja elküldeni a Kafka-ba írás el?tt a producer a séma regiszternek.

```
2019-04-02 13:30:44 DEBUG RestService:118 - Sending POST with input {"schema":"{\"type\":\"record\",\"name\":\"Employee\",\"namespace\":\"hu.
```

Ez avro schema-registry szintjén azt jelenti, hogy a **test-topic-key** subject-et nem fogja használni/létrehozni.

```
$ curl -X GET -H "Content-Type: application/vnd.schemaregistry.v1+json" http://192.168.42.42:8081/subjects/

["test-topic-value","test-topic-key","test1"]
```

Command line producers

https://docs.confluent.io/3.0.0/quickstart.html
A Confluent oldaláról letölthet? Kafka csomagban található parancssori kafka-avro producer és consumer is. Töltsük le a Confulent csomagot innen: https://www.confluent.io/download/

A **kafka-avro-console-producer** program a /bin mappában található. 4 paramétert kell kötelez?en kitöltenünk:

- broker-list: itt meg kell adni a Kafka broker URL-jét
- topic: meg kell adni a Kafka topic nevét, ahova írja az üzeneteket
- property: itt fel tudunk sorolni tetsz?leges paramétereket, nekünk itt kett?t kell kötelez?en megadni:
  - schema.registry.url: A séma regiszter elérhet?sége
  - value.schema: itt meg kell adni a használni kívánt sémát JSON formátumban (a Java producer esetén a séma bele van kódolva az Avro objektumokban, a példában az Employee objektum elején megtalálhatjuk a sémát. A sémát minden producer indulás elején felküldi a producer a séma regiszterbe, hogy ellen?rizze, hogy változott e vagy sem.)

Emlékezzünk rá, hogy az Employee séma az alábbi volt:

```
{"namespace": "hu.alerant.kafka.avro.message",
  "type": "record",  "name": "Employee",
    "fields": [
        {"name": "firstName", "type": "string"},
        {"name": "lastName", "type": "string"},
        {"name": "age",  "type": "int"},
        {"name": "phoneNumber",  "type": "string"}
    ]
}
```

Ezt majd meg kell adjuk egysoros alakban a **kafka-avro-console-producer** parancsban.

Note
A **kafka-avro-console-producer** parancsban a konkrét Avro üzenetet nem lehet megadni. Miután kiadtuk a parancsot, az input-on fogja várni, hogy bírjuk JSON formátumban a sémának megfelel? üzenetet. Minden egyes Enter leütésre megpróbálja elküldeni amit az stdIn-re beírtunk

Az Employee séma használata mellett a parancs az alábbi:

```
./kafka-avro-console-producer \
--broker-list kafka:29092 \
--topic test-topic \
--property schema.registry.url='http://schema-registry:8081' \
--property value.schema='{"namespace": "hu.alerant.kafka.avro.message", "type": "record",  "name": "Employee", "fields": [{"name": "firstName
```

Ekkor várni fogja hogy egy sorba bírjuk az els? átküldend? üzenetet JSON formában, ami megfelel a fenti sémának:

```
{"firstName": "Adam", "lastName": "Berki", "age":20, "phoneNumber": "123456"}
```

Ha bemásoltuk, akkor az Enter leütésével küldhetjük be az üzenetet. Ekkor a producer fel fogja küldeni a sémát a séma regiszterbe, majd az Avro segítségével serializálni fogja az üzenetet, majd a séma regisztert?l kapott séma ID-t és a bináris üzenetet rá fogja rakni a test-topic-ra.

Consumers

Command line consumer

A command line producer-el megegyez?en, szintén a Confluent oldaláról letölthet? Kafka csomagban találhatjuk meg a command line kafka-avro consumer-t.

A **kafka-avro-console-consumer** program a /bin mappában található. Használata nagyon hasonlít a producer-re, 3 kötelez? paramétere van:

- bootstrap-server: itt meg kell adni a Kafka broker URL-jét
- topic: meg kell adni a Kafka topic nevét, ahonnan olvassuk az üzeneteket
- property: itt fel tudunk sorolni tetsz?leges consumer paramétereket, itt adhatjuk meg a séma regiszter URL-jét: schema.registry.url

Mikor a consumer üzenetet kap, el fog menni a séma regiszterhez, hogy letöltse az üzenetben kapott séma ID-hez tartozó sémát. Ez alapján fogja deseralizálni az üzentet. Indítsuk el a consumert-t, amjd a command line producer segítségével küldjünk bele Employee üzeneteket. Láthatjuk majd, hogy JSON formátumban meg fogjuk kapni az eredeti üzenetet.

```
./kafka-avro-console-consumer \
--bootstrap-server kafka:29092 \
--topic test-topic \
--property schema.registry.url='http://schema-registry:8081'
...
{"firstName":"Adam","lastName":"Berki","age":20,"phoneNumber":"123456"}
```

Java consumer

A Properties map-ben a szokásos Kafka specifikus paramétereken felül meg kell adjuk a séma regiszter URL-jét és a séma használatára vonatkozó beállításokat.

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class.getName());
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class.getName());
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "true");
props.put(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "http://schema-registry:8081");
```

Séma specifikus consumer

Ha a KafkaAvroDeserializerConfig.**SPECIFIC_AVRO_READER_CONFIG** értéke igaz, akkor a választ egy el?re meghatározott objektum típusban fogjuk visszakapni, a példában ez lesz a **Employee.java**

```
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "true");
```

Mikor a consumer-t példányosítjuk, már ott meg kell adni, hogy mi az az Avro típus, amit válaszként várunk. Majd mikor elkérjük a consumer-t?l az üzenetet, akkor is pontosan meg kell adni a típust.

```
Consumer<Long, Employee> consumer = createConsumer();
...
final ConsumerRecords<Long, Employee> records = consumer.poll(Duration.ofMillis(100));
```

Ez a teljes consumer:

```
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;
import java.util.stream.IntStream;

import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import hu.alerant.kafka.avro.message.Employee;
import io.confluent.kafka.serializers.KafkaAvroDeserializer;
import io.confluent.kafka.serializers.KafkaAvroDeserializerConfig;

public class AvroConsumer {

private final static String BOOTSTRAP_SERVERS = "kafka:29092";
private final static String TOPIC = "test-topic";

private static Consumer<Long, Employee> createConsumer() {
Properties props = new Properties();
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, BOOTSTRAP_SERVERS);
```

```
props.put(ConsumerConfig.GROUP_ID_CONFIG, "KafkaExampleAvroConsumer");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class.getName());

props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class.getName());
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "true");

props.put(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "http://schema-registry:8081");
return new KafkaConsumer<>(props);
}

public static void main(String... args) {
final Consumer<Long, Employee> consumer = createConsumer();

consumer.subscribe(Collections.singletonList(TOPIC));

IntStream.range(1, 100).forEach(index -> {
final ConsumerRecords<Long, Employee> records = consumer.poll(Duration.ofMillis(100));
if (records.count() == 0) {
System.out.println("None found");
} else
records.forEach(record -> {
Employee employeeRecord = record.value();
System.out.printf("%s %d %d %s \n", record.topic(), record.partition(), record.offset(),
employeeRecord);
});
});


consumer.close();
}
}
```

Warning
A ***org.apache.kafka.clients.consumer.KafkaConsumer.poll(long)*** már deprecated. Helyette a **KafkaConsumer.poll(Duration)** metódust kell használni

Küldjünk üzeneteket a test-topic-ra. Az üzenetek csak az Employee sémára illeszked? objektumok lehetnek.

```
2019-04-02 12:25:04 DEBUG AbstractCoordinator:822 - [Consumer clientId=consumer-1, groupId=KafkaExampleAvroConsumer] Received successful Hear
test-topic 0 1 {"firstName": "Bob", "lastName": "Jones", "age": 35, "phoneNumber": ""}
```

Generikus consumer

Ha a KafkaAvroDeserializerConfig.**SPECIFIC_AVRO_READER_CONFIG** értéke hamis, akkor a választ a válasz paroszlására a **GenericRecord** nev? általános célú objektumot kell használni, amib?l extra munkával lehet csak kinyerni az eredeti objektum mez?it, cserélbe nem kell séma specifikus consumer-t írni.

```
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "false");
```

Mikor a consumer-t példányosítjuk, meg kell adni a GenericRecord típust. Majd mikor elkérjük a consumer-t?l az üzenetet, akkor is a **GenericRecord**-t kell megadni:

```
final Consumer<Long, GenericRecord> consumer = createConsumer();
...
ConsumerRecords<Long, GenericRecord> records = consumer.poll(Duration.ofMillis(100));
```

Ez a teljes consumer:

```
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import io.confluent.kafka.serializers.KafkaAvroDeserializer;
import io.confluent.kafka.serializers.KafkaAvroDeserializerConfig;

public class AvroConsumerGeneric {

private final static String BOOTSTRAP_SERVERS = "kafka:29092";
private final static String TOPIC = "test-topic";

private static Consumer<Long, GenericRecord> createConsumer() {
Properties props = new Properties();
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, BOOTSTRAP_SERVERS);
props.put(ConsumerConfig.GROUP_ID_CONFIG, "KafkaExampleAvroConsumer");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
org.apache.kafka.common.serialization.LongDeserializer.class);
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class.getName());
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "false");
props.put(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG, "http://schema-registry:8081");
return new KafkaConsumer<>(props);
}

public static void main(String... args) {
final Consumer<Long, GenericRecord> consumer = createConsumer();
consumer.subscribe(Collections.singletonList(TOPIC));

try {
```

```
while (true) {
ConsumerRecords<Long, GenericRecord> records = consumer.poll(Duration.ofMillis(100));

for (ConsumerRecord<Long, GenericRecord> record : records) {
GenericRecord valueGr = record.value();
System.out.printf("offset = %d, key = %s, value = %s \n", record.offset(), record.key(),
valueGr.toString());
}
}
} finally {
consumer.close();
}
}
}
```

Láthatjuk, hogy a **GenericRecord** példányban ott van a producer által küldött JSON:

```
offset = 8, key = 123456778, value = {"firstName": "Bob", "lastName": "Jones", "age": 35, "phoneNumber": ""}
```

Partition keys

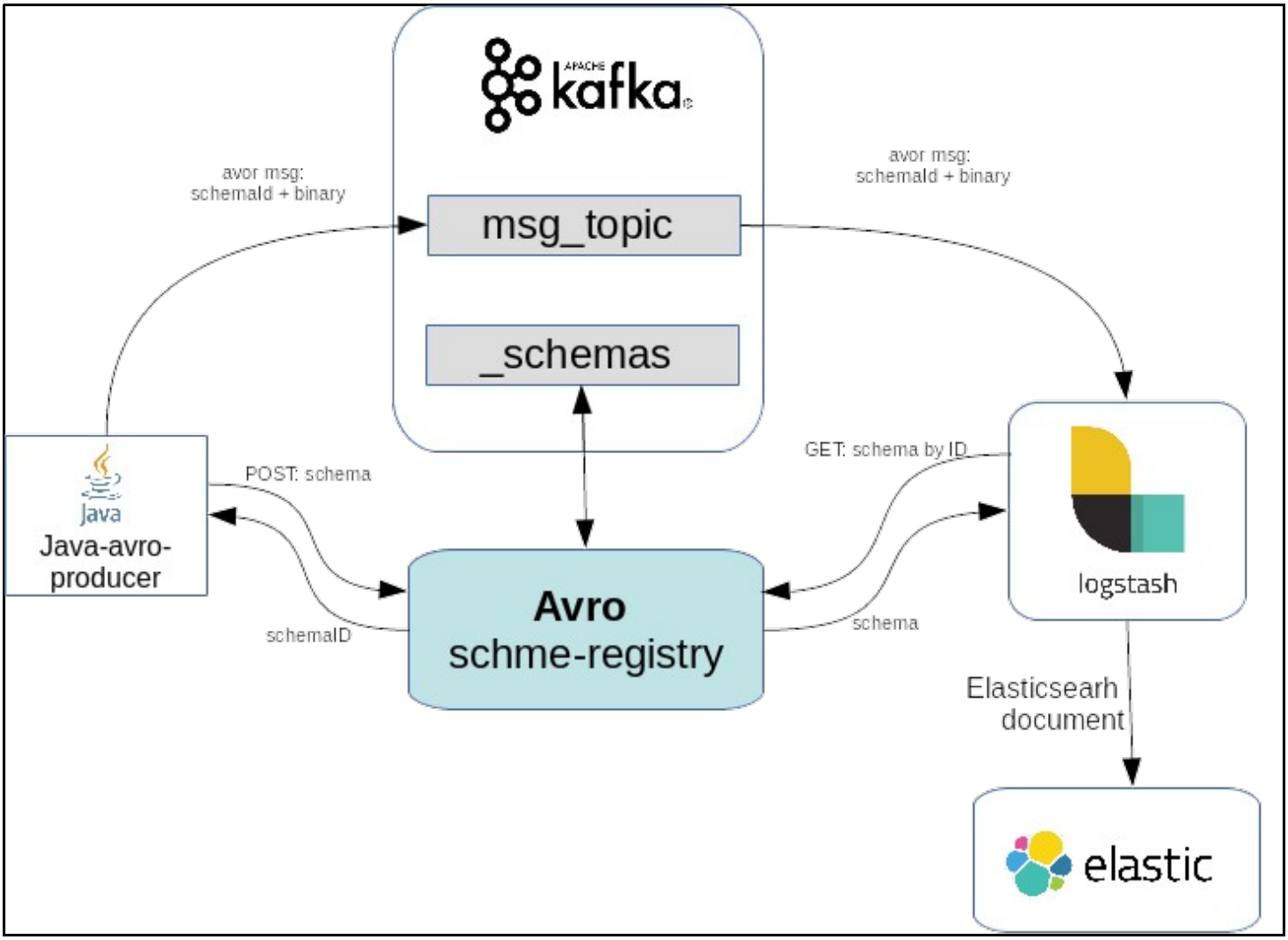
Akárcsak a producer esetén, a consumer-ben is használható nem Avro-s kulcs. Lényeg, hogy a consumer-ben ugyan azt a kulcs szerializációs eljárást kell használni, mint a producer-ben:

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, org.apache.kafka.common.serialization.LongDeserializer.class);
```

Logstash consumer

Hasznos linkek: https://www.elastic.co/guide/en/logstash/6.5/plugins-codecs-avro.html
https://github.com/revpoint/logstash-codec-avro_schema_registry
https://www.elastic.co/guide/en/logstash/current/plugins-codecs-avro.html

A logstash-t tehetjük a Kafka elé és mögé is. Ha a Kafka mögé tesszük, pl. a Kafka és az Elasticsearh közé, akkor a logstash-nek kell elvégezni az Avro deszeralizációt az Elastichsearh-be való írás el?tt. Az avro üzenetek feldolgozására nem képes önállóan a Logstash, szükség van egy megfelel? input avro-kafka-logstash plugin-re, ami el tudja végezni a deserializációt.



Sajnos az alap docker logstash image nem tartalmazza semmilyen avro támogatást. A plugin-ek listáját a **bin/logstash-plugin list** paranccsal ellen?rizhetjük. Láthatjuk, hogy az alap docker image nem tartalmazza az avro plugin-t.

```
# docker run -it docker.elastic.co/logstash/logstash:6.6.2 bin/logstash-plugin list
logstash-codec-cef
logstash-codec-collectd
logstash-codec-dots
...
```

Két lehet?ségünk van. Egyik megoldás, hogy készítünk egy új image-et a logstash:6.6.2 image-b?l kiindulva. Új plugin-t a **bin/logstash-plugin install** paranccsal installálhatunk. Ezt betehetjük egy Dockerfile-ba. A nehézség, hogy a swarm node-okra el kell juttatni a módosított image-t vagy saját repot kell használni. A másik lehet?ség, hogy a dockerhub-on keresünk olyan image-et amibe már installáltak avro plugin-t.

```
bin/logstash-plugin install logstash-codec-avro
```

logstash avro plugin

Két AVRO codek érhet? a logstash-hez:

1. https://www.elastic.co/guide/en/logstash/6.5/plugins-codecs-avro.html
Ez a codek nem kapcsolódik a séma regiszterhez, el?re meghatározott séma fájl alapján tud dekódolni, amit a fájlrendszerben oda kell másoljunk logstash-be. Ha változik a séma, akkor kézzel ki kell cserélni.

2. https://github.com/revpoint/logstash-codec-avro_schema_registry
Ez a codek tud csatlakozni a séma regiszterhez. Tehát nem kell kézzel karbantartsuk a sémákat.

```
bin/logstash-plugin install logstash-codec-avro_schema_registry
```

A docker-hub-on elérhet? két logstash image is, ami tartalmazza a logstash-codec-avro_shcema_registry plugin-t:

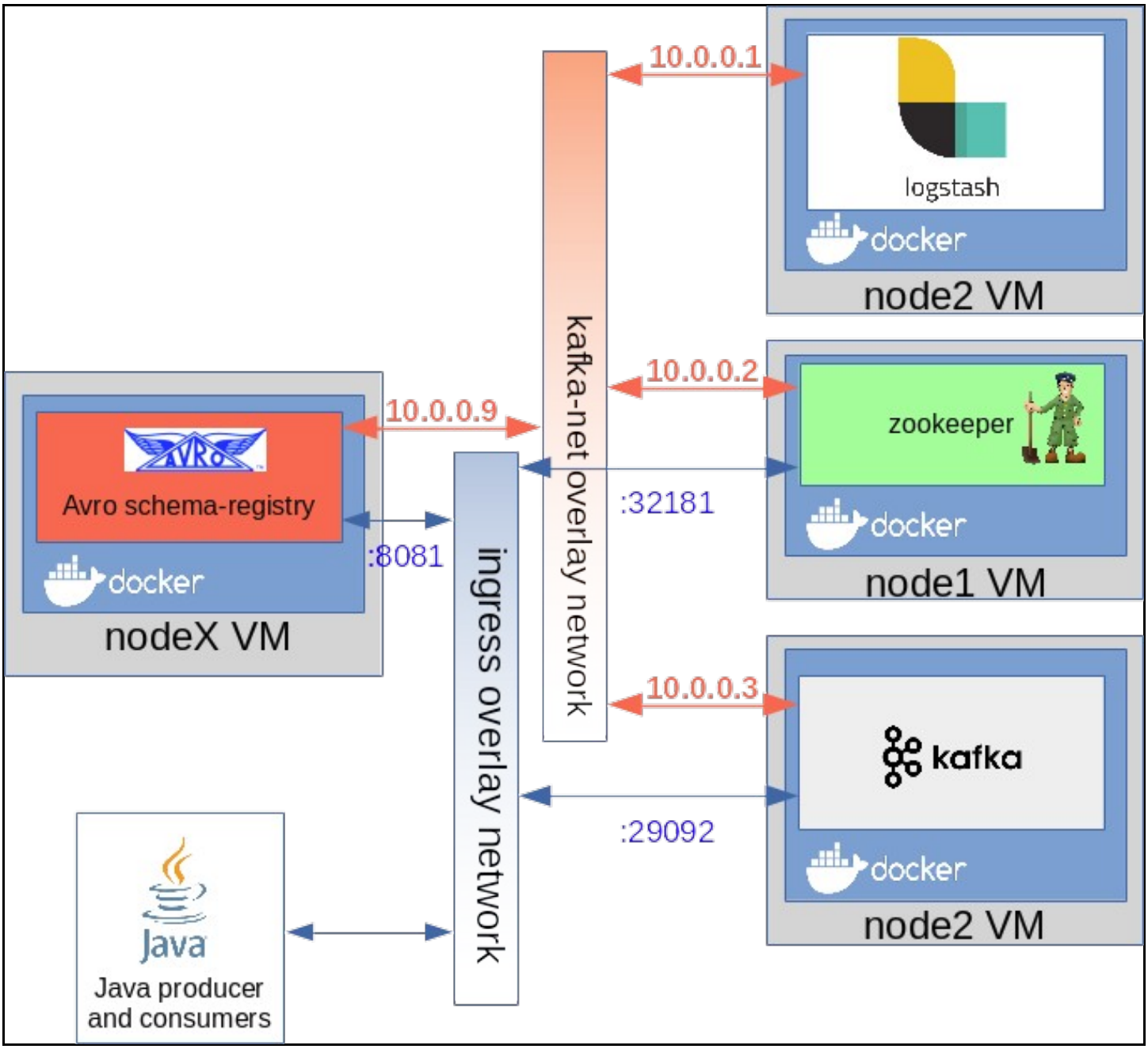
- https://hub.docker.com/r/rokasovo/logstash-avro2 --> ocker pull rokasovo/logstash-avro2
- https://hub.docker.com/r/slavirok/logstash-avro --> docker pull slavirok/logstash-avro

Listázzuk ki a második image-ben a plugine-eket. Láthatjuk, hogy az avro plugin köztük van.

```
# docker run -it rokasovo/logstash-avro2 bin/logstash-plugin list
logstash-codec-avro_schema_registry
logstash-codec-cef
...
```

Swarm architektúra

A swarm architektúrát b?víteni fogjuk a **rokasovo/logstash-avro2 logstash** komponenssel. A logstash a bels? **kafka-net** overlay hálózaton fogja elérni a schema-registry-t. Az Elasticsearh-öt már nem tesszük be a swarm stack-be, a logstash által feldolgozott üzeneteket csak ki fogjuk loggolni:



A logstash konfigurációját küls? volume-ként fogjuk felcsatolni a netshare plugin használatával (részletek itt: https://wiki.berki.org/index.php/Docker_volume_orchestration)

```
version: '3.2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.1.2
    networks:
      - kafka-net
    ports:
      - "32181:32181"
    deploy:
      placement:
        constraints:
          - node.role == worker
    environment:
      ZOOKEEPER_CLIENT_PORT: 32181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_SYNC_LIMIT: 2
  kafka:
    image: confluentinc/cp-kafka:5.1.2
    networks:
      - kafka-net
    ports:
      - target: 29092
        published: 29092
        protocol: tcp
    deploy:
      placement:
        constraints:
          - node.role == worker
    environment:
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:32181"
      KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka:29092"
```

```
      KAFKA_BROKER_ID: 2
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  schemaregistry:
    image: confluentinc/cp-schema-registry:5.1.2
    networks:
      - kafka-net
    ports:
      - target: 8081
        published: 8081
        protocol: tcp
    deploy:
      placement:
        constraints:
          - node.role == worker
    environment:
      SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: "zookeeper:32181"
      SCHEMA_REGISTRY_HOST_NAME: "schemaregistry"
      SCHEMA_REGISTRY_DEBUG: "true"
  logstash:
    image: rokasovo/logstash-avro2
    networks:
      - kafka-net
    ports:
      - "51415:51415"
    environment:
      LOGSPOUT: "ignore"
      XPACK_MONITORING_ENABLED: "false"
    volumes:
      - "logstash-conf:/usr/share/logstash/pipeline"
    deploy:
      placement:
        constraints:
          - node.role == worker
      restart_policy:
        condition: on-failure
      resources:
        reservations:
          memory: 100m
networks:
  kafka-net:
    driver: overlay
volumes:
  logstash-conf:
    driver: nfs
    driver_opts:
      share: 192.168.42.1:/home/adam/dockerStore/logstash/config/
```

Logstash config

A logstash image-en belül a konfigurációs fájl itt található: /usr/share/logstash/pipeline/**logstash.conf**. Ide kell felcsatolni a küls? konfigurációs fájlt. A kafka input-ban a codec-nek meg kell adni a **avro_schema_registry plugin**-t, amit a **rokasovo**/**logstash-avro2** image már tartalmaz. Az **endpoint** paraméterben kell megadni a schema-registry url-jét. Fontos, hogy itt a bels?, kafka-net overlay hálózati nevet adjuk meg, ami megegyezik a stack fájlban a service nevével. Ugyanis a service nevére egy stack-en belül a docker névfeloldást végez. Valamiért a deserializációs osztálynak a **ByteArrayDeserializer** osztályt kell megadni, nem a **KafkaAvroDeserializer** osztályt (A KafkaAvroDeserializer-t nem tartalmazza az avro input plugin) Az output-ban egyenl?re nem írjuk be Elasticsearch-be az üzeneteket, csak kiírjuk a log-ba.

```
input {
    kafka {
        codec => avro_schema_registry {
            endpoint => "http://schemaregistry:8081"
        }
        decorate_events => true
        value_deserializer_class => "org.apache.kafka.common.serialization.ByteArrayDeserializer"

        topics => [
            "test-topic"
        ]

        bootstrap_servers => "kafka:29092"
        group_id => "AvroConsumerGroupId"
        client_id => "AvroConsumerClientId"

    }
}

output {

  stdout {
    codec => rubydebug
  }
}
```

Ha a helyére tettük a konfigurációs fájlt, akkor indítsuk el **docker run** paranccsal lokálisan felcsatolva a /**usr**/**share**/**logstash**/**pipeline** mappába a konfigurációs fájlt, hogy ki tudjuk külön próbálni, hogy a konfiguráció megfelel? e. Persze a Kafka-hoz nem fog tudni csatlakozni, de a szintaktikai hibákat tudjuk ellen?rizni.

```
# docker run -d --name logstash --mount type=bind,source=/home/adam/dockerStore/logstash/config/,target=/usr/share/logstash/pipeline rokasovo
```

Futtatás

Telepítsük fel a docker swarm stack-et:

```
# docker stack deploy -c confluent_swarm_logstash.yaml  confluent
Creating network confluent_kafka-net
Creating service confluent_logstash
Creating service confluent_zookeeper
Creating service confluent_kafka
Creating service confluent_schemaregistry
```

Majd nézzük bele a logstash service logjába. Látnunk kell, hogy hozzá tudod csatlakozni a **test-topic** nev? topichoz. A séma regiszterhez csak az üzenet deszerializációja közben fog csatlakozni, ezért nem láthatjuk még a logban.

```
# docker service logs -f confluent_logstash
...
[INFO ]Kafka version : 1.0.0
[INFO ]Kafka commitId : aaa7af6d4a11b29d
[INFO ][Consumer clientId=AvroConsumerClientId-0, groupId=AvroConsumerGroupId] Discovered coordinator kafka:29092 (id: 2147483645 rack: null)
[INFO ][Consumer clientId=AvroConsumerClientId-0, groupId=AvroConsumerGroupId] Revoking previously assigned partitions []
[INFO ][Consumer clientId=AvroConsumerClientId-0, groupId=AvroConsumerGroupId] (Re-)joining group
[INFO ][Consumer clientId=AvroConsumerClientId-0, groupId=AvroConsumerGroupId] Successfully joined group with generation 5
[INFO ][Consumer clientId=AvroConsumerClientId-0, groupId=AvroConsumerGroupId] Setting newly assigned partitions [test-topic-0]
```

Majd indítsuk el a **Java avro-kafak producer** fejezetben leírt java producert, ami egy egy Employee objektumot fog beküldeni a test-topic-ba. Emlékezzünk rá, hogy az Employee objektum sémája az alábbi:

```
{"namespace": "hu.alerant.kafka.avro.message",
  "type": "record",  "name": "Employee",
    "fields": [
        {"name": "firstName", "type": "string"},
        {"name": "lastName", "type": "string"},
        {"name": "age",  "type": "int"},
        {"name": "phoneNumber",  "type": "string"}
    ]
}
```

A java producer-ben az Employee objektum példányosítása az alábbi:

```
...
        Producer<Long, Employee> producer = createProducer();
        Employee bob = Employee.newBuilder().setAge(35)
                .setFirstName("Bob")
                .setLastName("Jones")
                .setPhoneNumber("")
                .build();
...
```

Miután a producer beküldte az avro üzenetet a Kafka test-topic-ba, a logstash logban meg kell jelenjen az alábbi üzenet:

```
    | {
    |         "@version" => "1",
    |              "age" => 35,
    |       "@timestamp" => 2019-04-19T10:39:30.266Z,
    |     "phoneNumber" => "",
    |       "firstName" => "Bob",
    |        "lastName" => "Jones"
    | }
```

Láthatjuk, hogy a logstash-avro plugin kiegészítette két meta adattal az üzenetet:

- @timestamp
- @version

A fenti konfig fájlban, az output egyszer? módosításával Elasticsarch-be írható az adat.