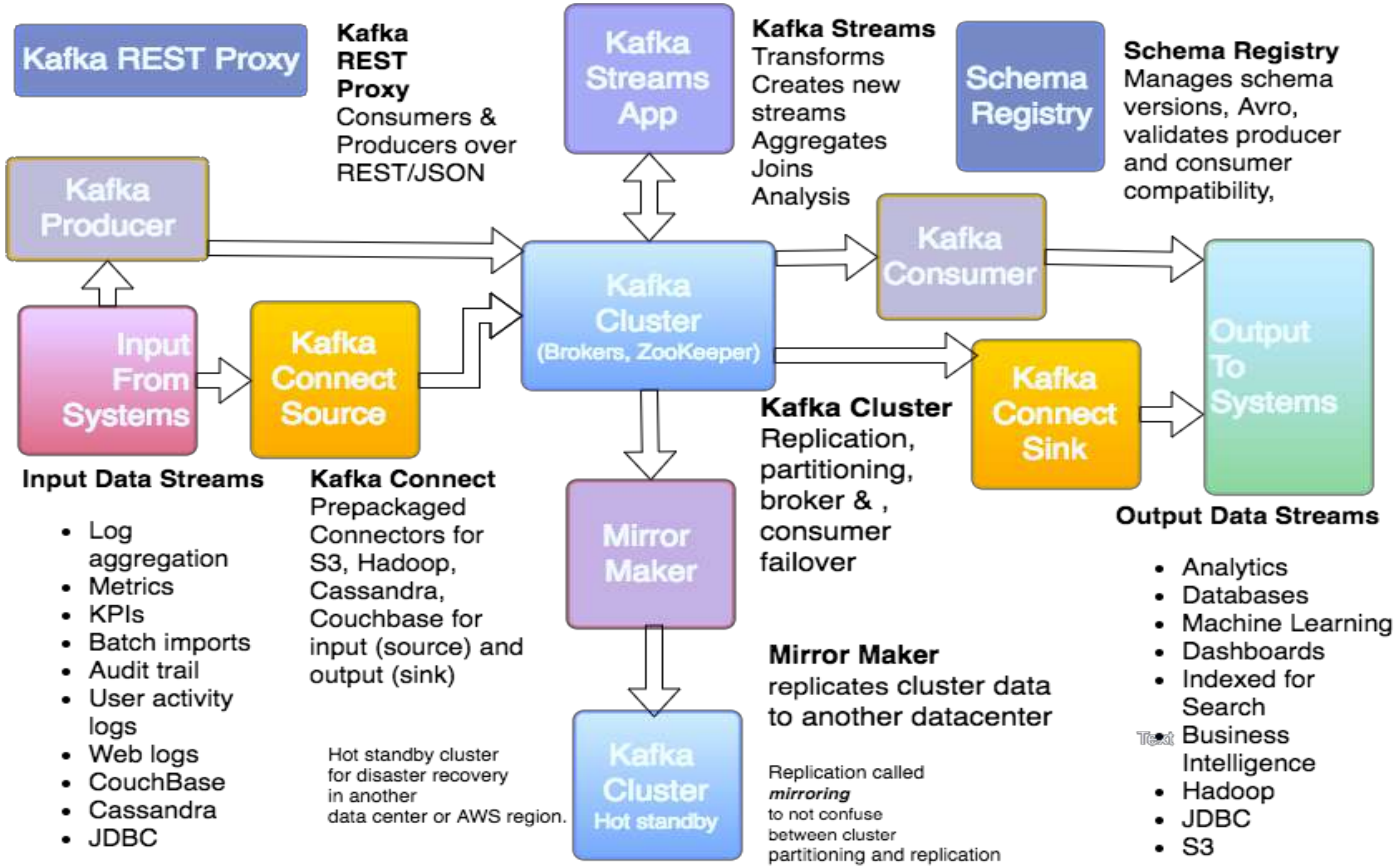
Kafka Ecosystem

Kafka Universe

Tos



Kafka REST Proxy
Consumers & Producers over REST/JSON

Kafka Streams
Transforms
Creates new streams
Aggregates
Joins
Analysis

Schema Registry
Manages schema versions, Avro, validates producer and consumer compatibility,

Kafka Cluster
Replication, partitioning, broker & , consumer failover

Input Data Streams

- Log aggregation
- Metrics
- KPIs
- Batch imports
- Audit trail
- User activity logs
- Web logs
- CouchBase
- Cassandra
- JDBC

Kafka Connect
Prepackaged Connectors for S3, Hadoop, Cassandra, Couchbase for input (source) and output (sink)

Hot standby cluster for disaster recovery in another data center or AWS region.

Mirror Maker
replicates cluster data to another datacenter

Replication called ***mirroring*** to not confuse between cluster partitioning and replication

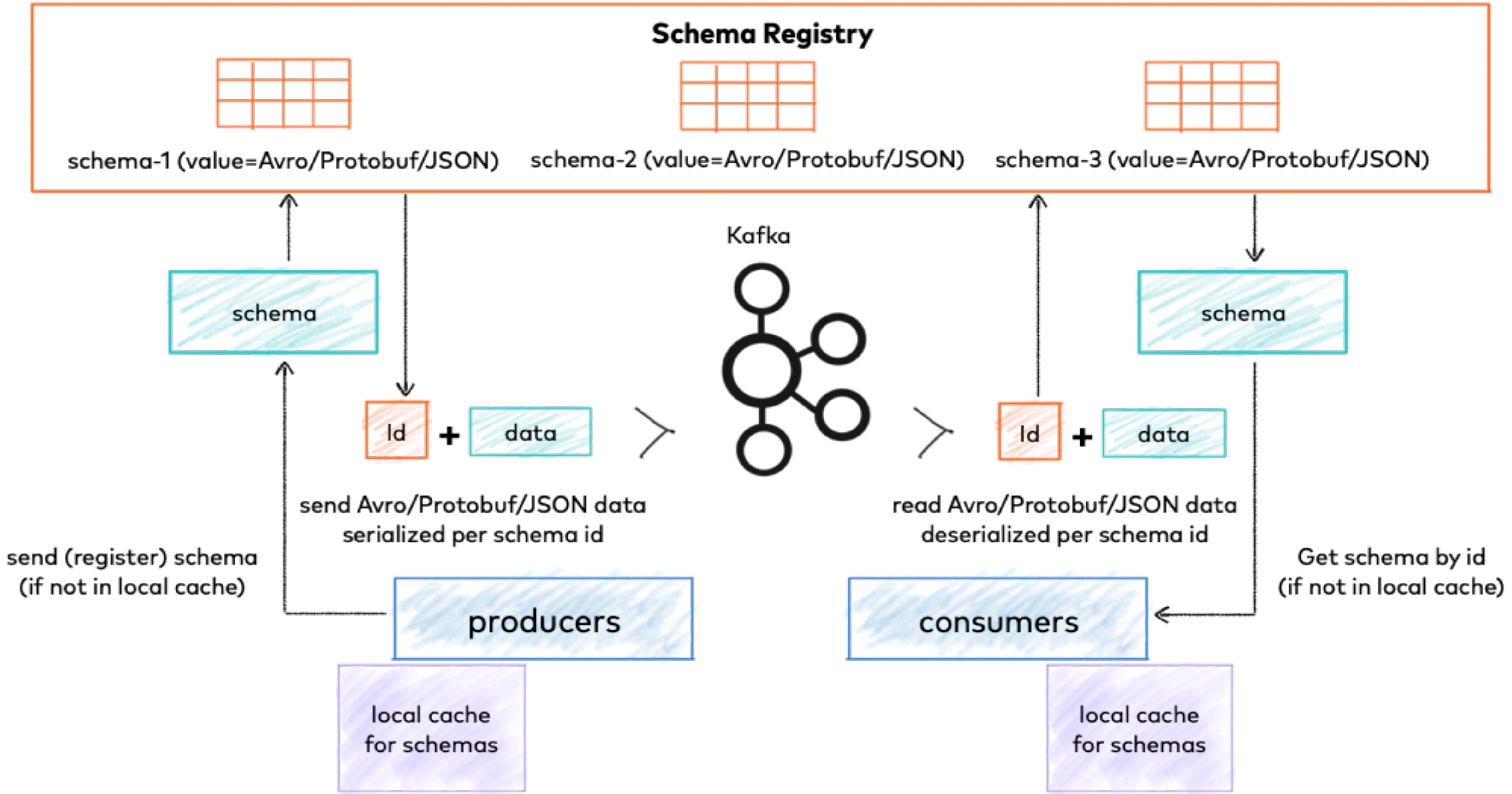
Output Data Streams

- Analytics
- Databases
- Machine Learning
- Dashboards
- Indexed for Search
- Business Intelligence
- Hadoop
- JDBC
- S3

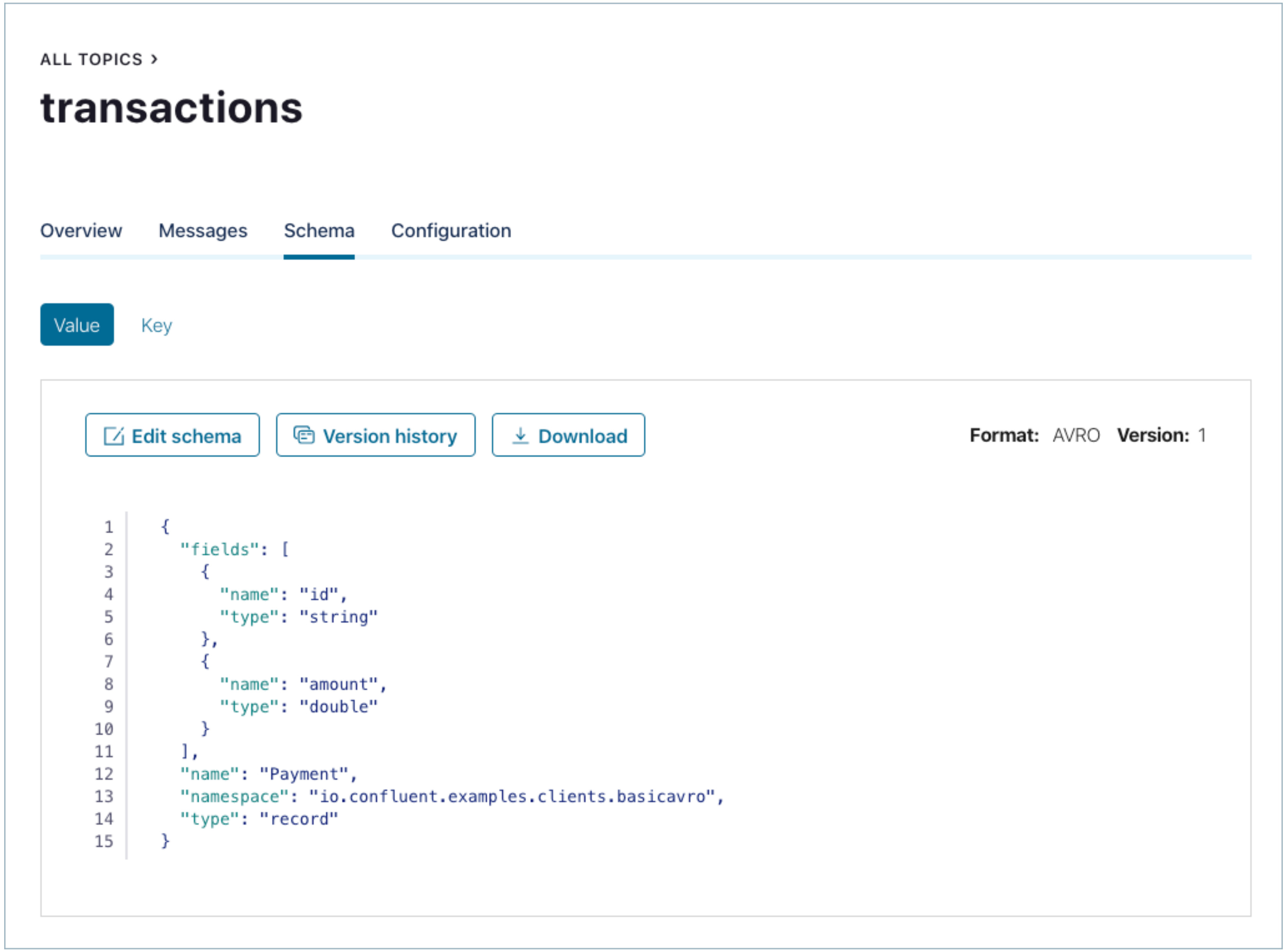
Tos

Schema Registry

Confluent Schema Registry


Schema Registry
schema-1 (value=Avro/Protobuf/JSON)
schema-2 (value=Avro/Protobuf/JSON)
schema-3 (value=Avro/Protobuf/JSON)
Kafka
schema
schema
Id
+
data
Id
+
data
send Avro/Protobuf/JSON data serialized per schema id
read Avro/Protobuf/JSON data deserialized per schema id
send (register) schema (if not in local cache)
Get schema by id (if not in local cache)
producers
consumers
local cache for schemas
local cache for schemas

Confluent Schema Registry - **Single Primary Architecture**



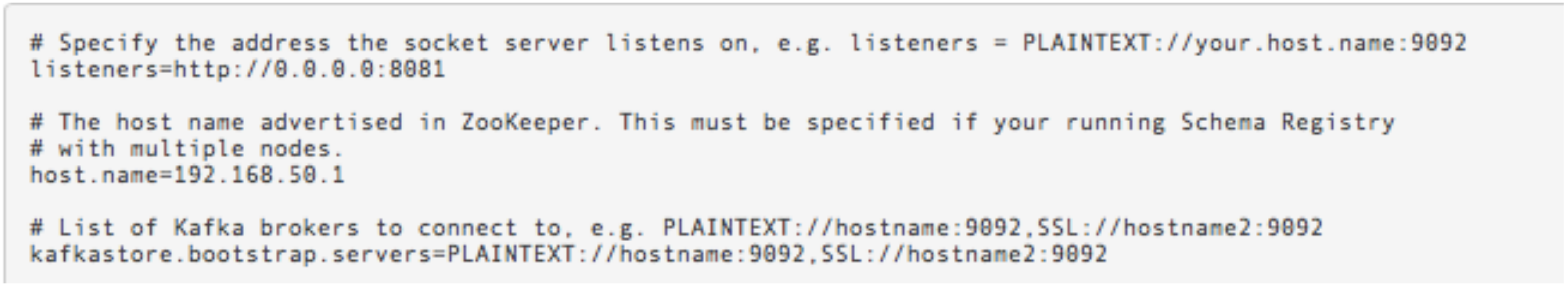
transactions

Format: AVRO **Version:** 1

```
{
  "fields": [
    {
      "name": "id",
      "type": "string"
    },
    {
      "name": "amount",
      "type": "double"
    }
  ],
  "name": "Payment",
  "namespace": "io.confluent.examples.clients.basicavro",
  "type": "record"
}
```

Confluent Schema Registry - **Single Primary Architecture**

etc/schema-registry/schema-registry.properties

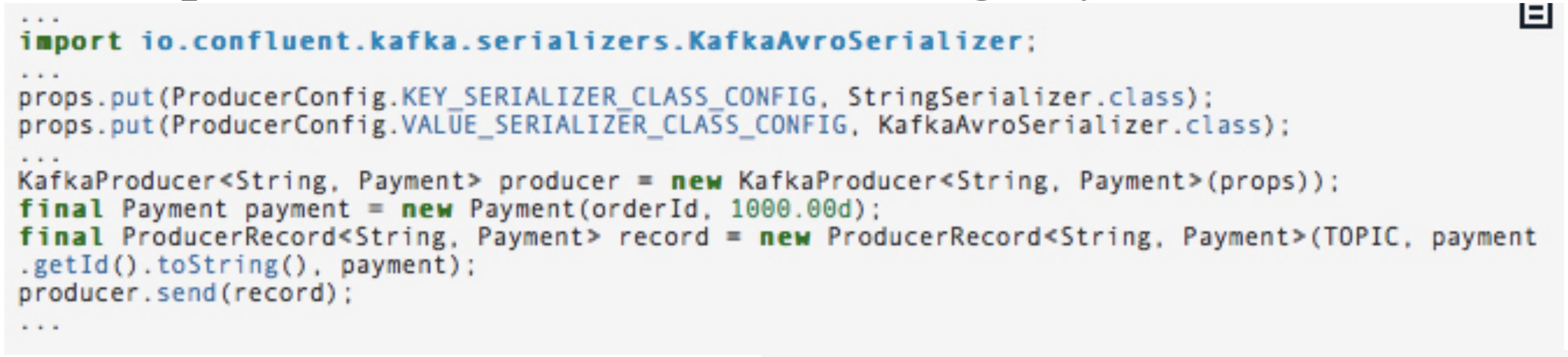


```
# Specify the address the socket server listens on, e.g. listeners = PLAINTEXT://your.host.name:9092
listeners=http://0.0.0.0:8081

# The host name advertised in ZooKeeper. This must be specified if your running Schema Registry
# with multiple nodes.
host.name=192.168.50.1

# List of Kafka brokers to connect to, e.g. PLAINTEXT://hostname:9092,SSL://hostname2:9092
kafkastore.bootstrap.servers=PLAINTEXT://hostname:9092,SSL://hostname2:9092
```

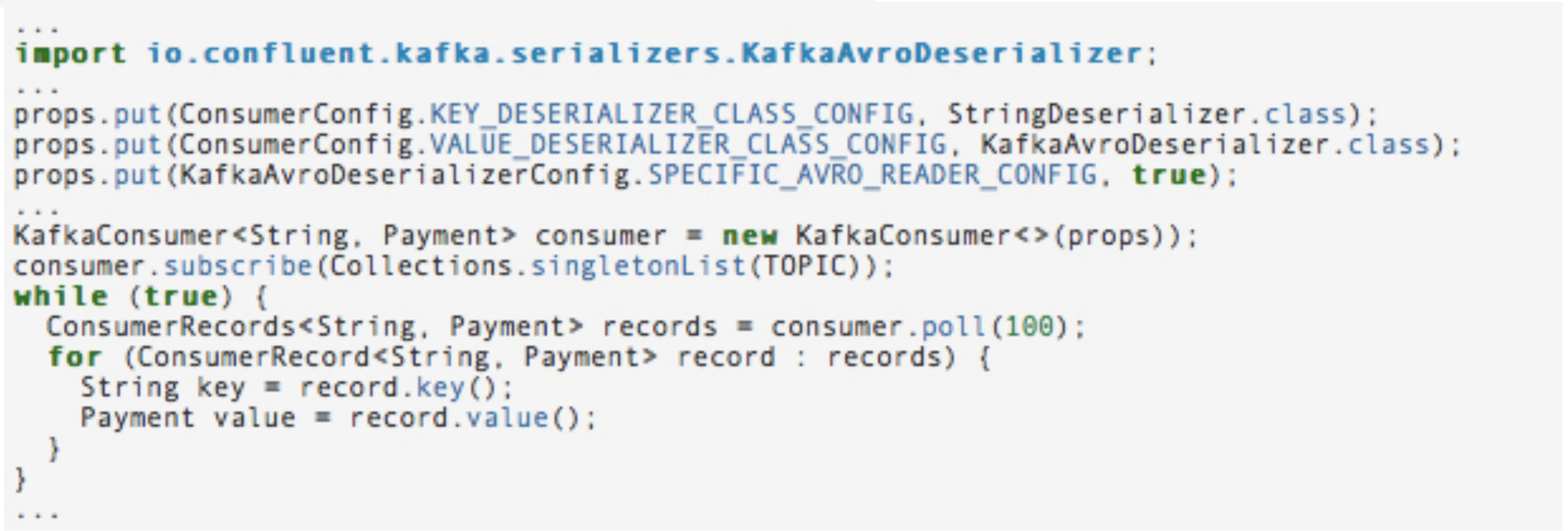
Configuring Avro

Kafka applications using Avro data and Schema Registry need to specify at least two configuration parameters:

- Avro serializer or deserializer
- Properties to connect to Schema Registry



```
...
import io.confluent.kafka.serializers.KafkaAvroSerializer;
...
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, KafkaAvroSerializer.class);
...
KafkaProducer<String, Payment> producer = new KafkaProducer<String, Payment>(props));
final Payment payment = new Payment(orderId, 1000.00d);
final ProducerRecord<String, Payment> record = new ProducerRecord<String, Payment>(TOPIC, payment
.getId().toString(), payment);
producer.send(record);
...
```



```
...
import io.confluent.kafka.serializers.KafkaAvroDeserializer;
...
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, KafkaAvroDeserializer.class);
props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, true);
...
KafkaConsumer<String, Payment> consumer = new KafkaConsumer<>(props));
consumer.subscribe(Collections.singletonList(TOPIC));
while (true) {
  ConsumerRecords<String, Payment> records = consumer.poll(100);
  for (ConsumerRecord<String, Payment> record : records) {
    String key = record.key();
    Payment value = record.value();
  }
}
...
```