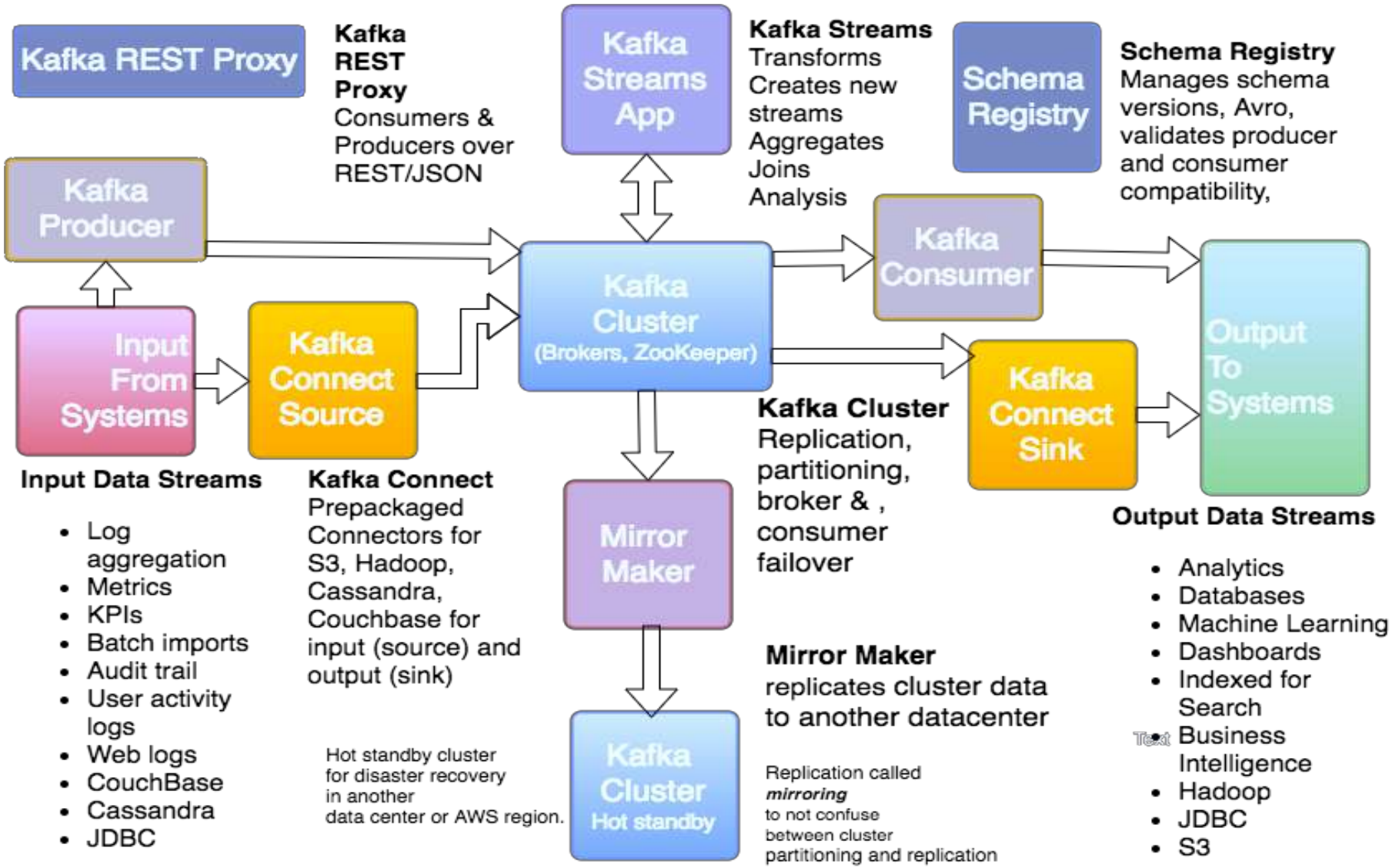
Kafka Ecosystem

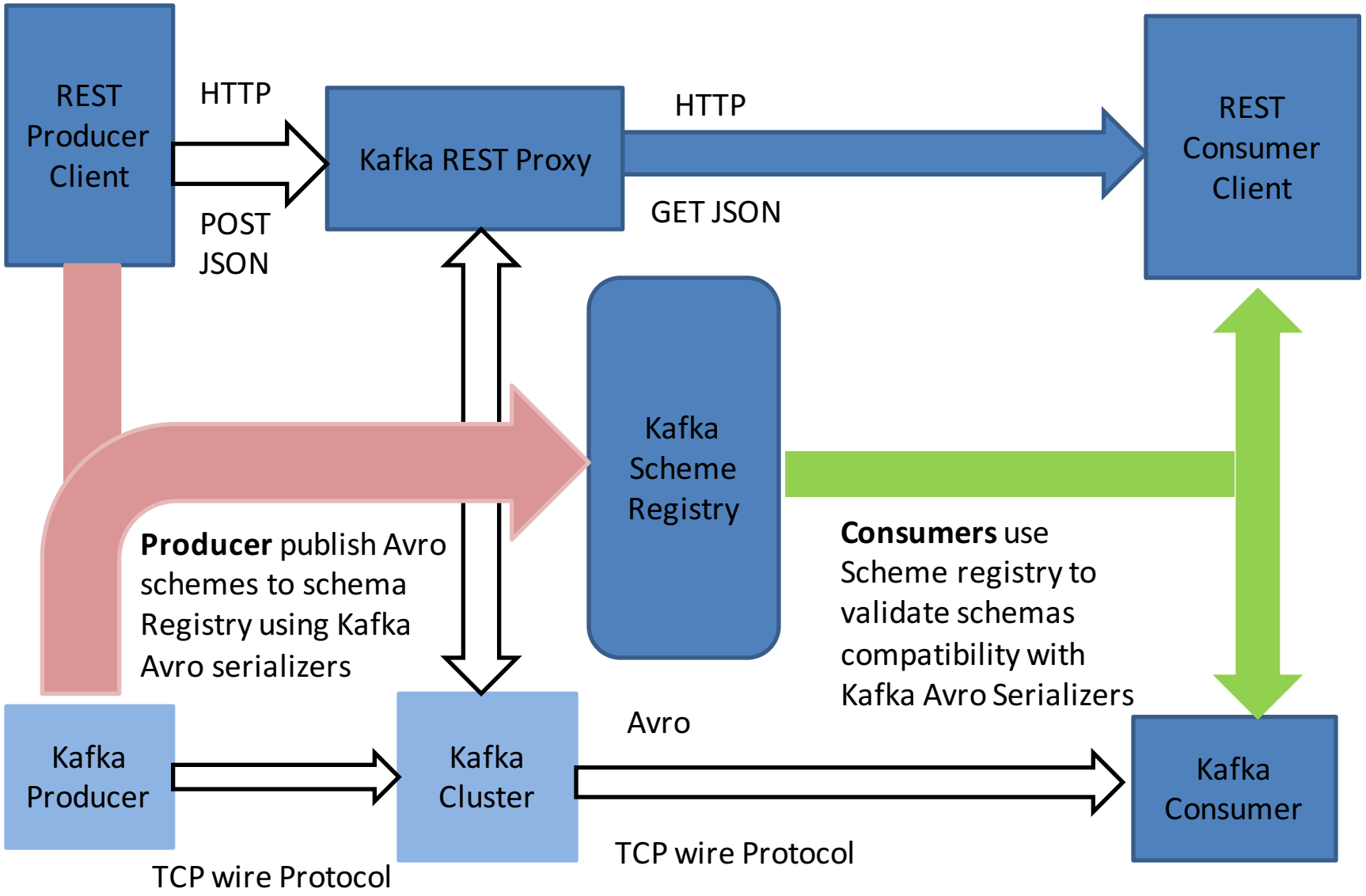
Kafka Universe

Tos



Kafka REST Proxy and Schema Registry

Tos

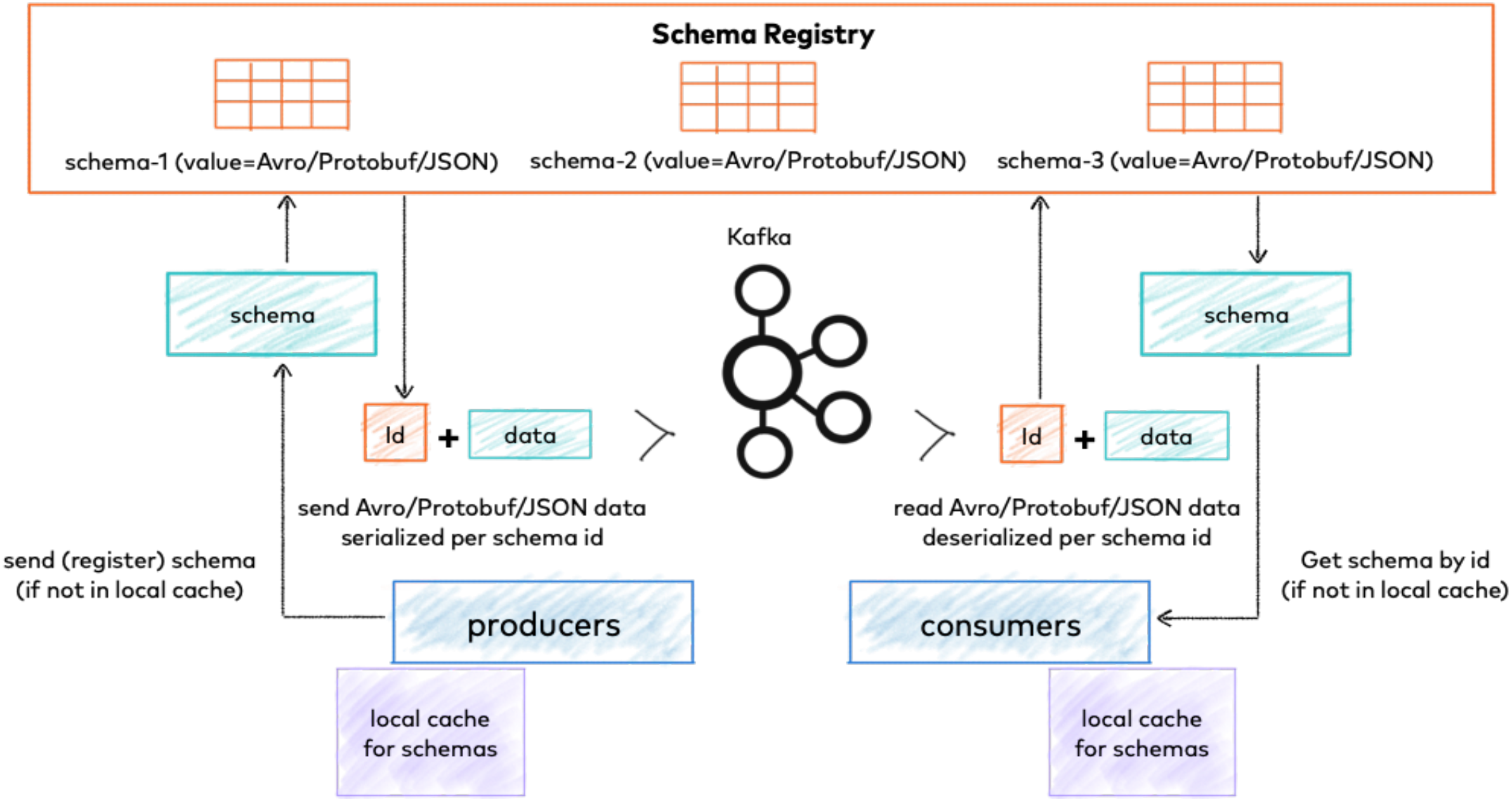


Schema Registry

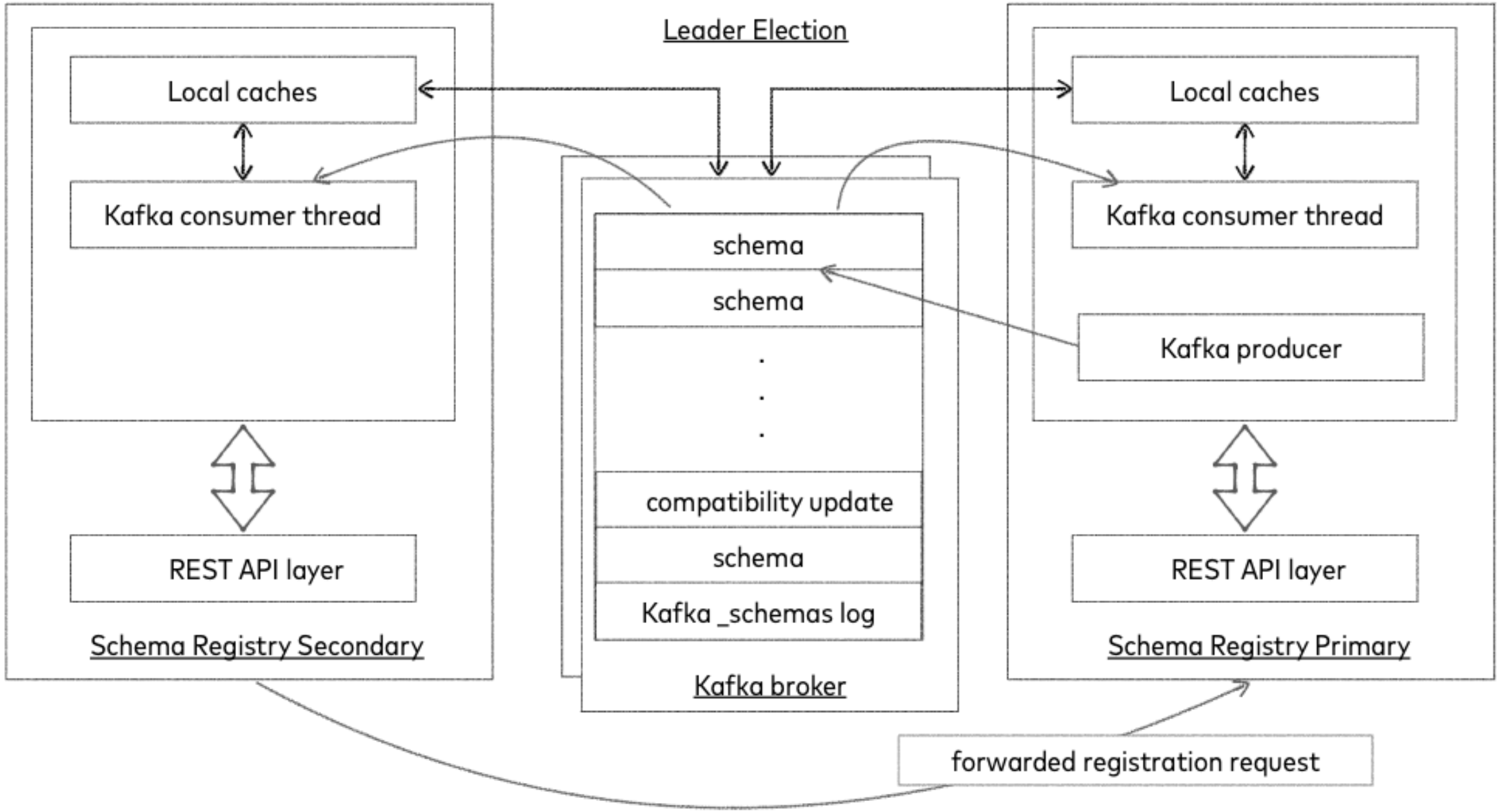
Confluent Schema Registry

Confluent Schema Registry provides a serving layer for your metadata.
It provides a RESTful interface for storing and retrieving your Avro®, JSON Schema, and Protobufschemas.
It stores a versioned history of all schemas based on a specified subject name strategy, provides multiple compatibility settings and allows evolution of schemas according to the configured compatibility settings and expanded support for these schema types.
It provides serializers that plug into Apache Kafka® clients that handle schema storage and retrieval for Kafka messages that are sent in any of the supported formats.

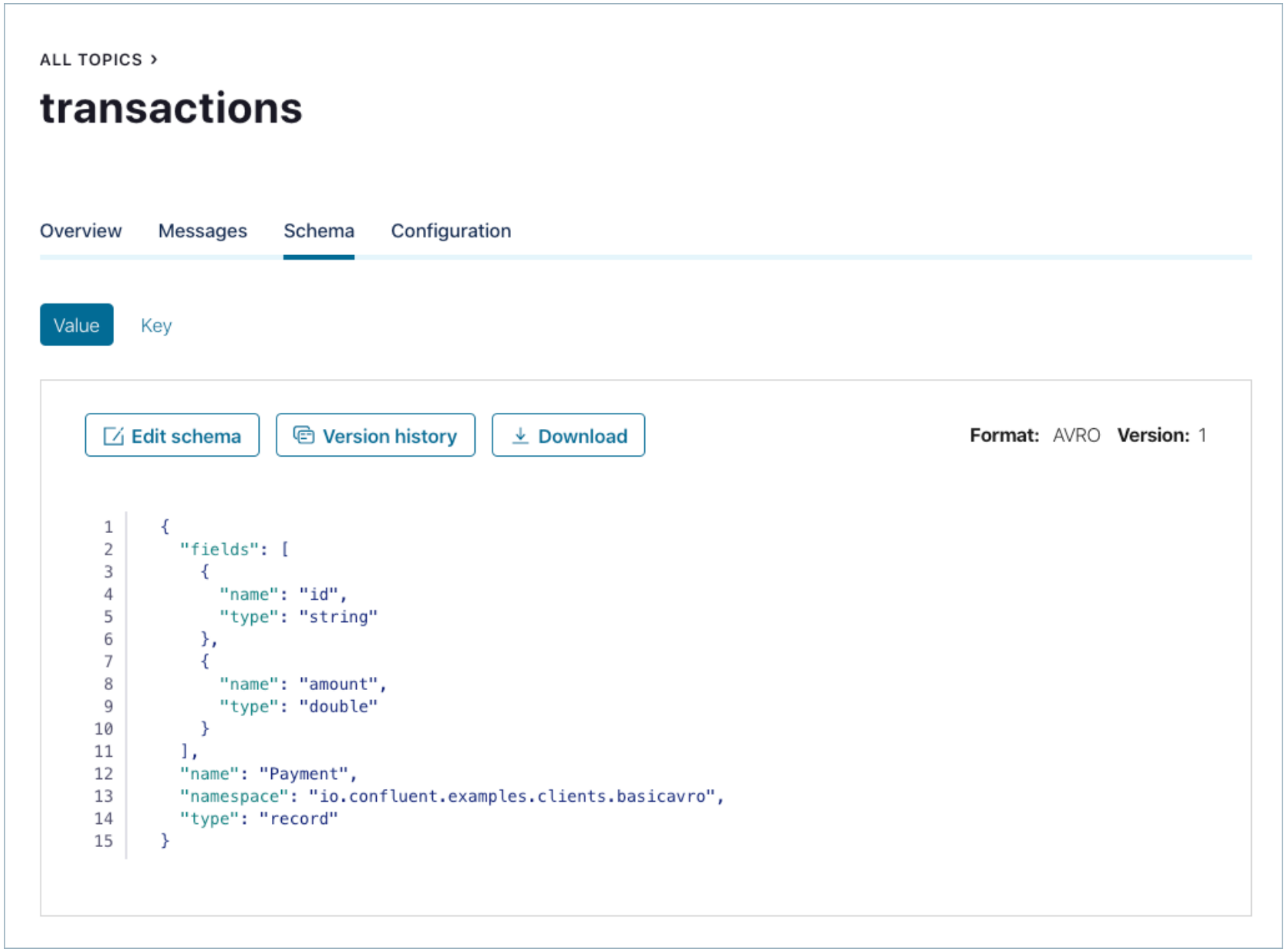
Confluent Schema Registry



Confluent Schema Registry - **Single Primary Architecture**



Confluent Schema Registry - **Single Primary Architecture**



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();
  }
}
...
```

Lab : Schema Registry - Manage Schemas for Topics