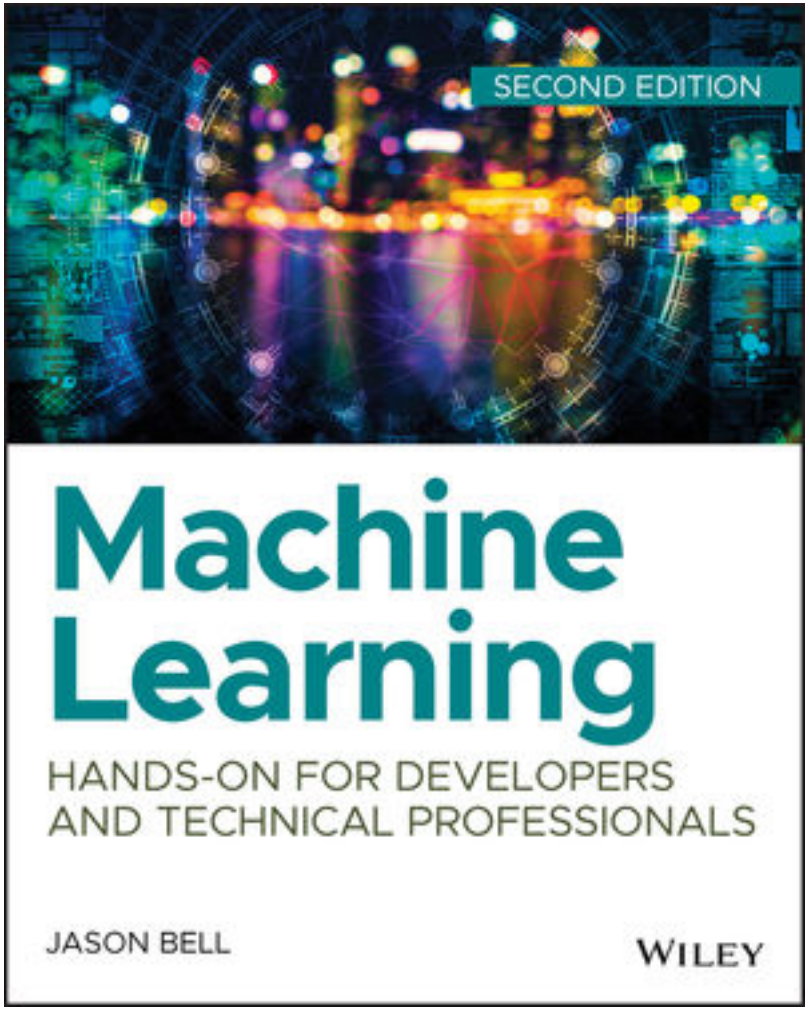
An Introduction to Kafka

That awkward slide….

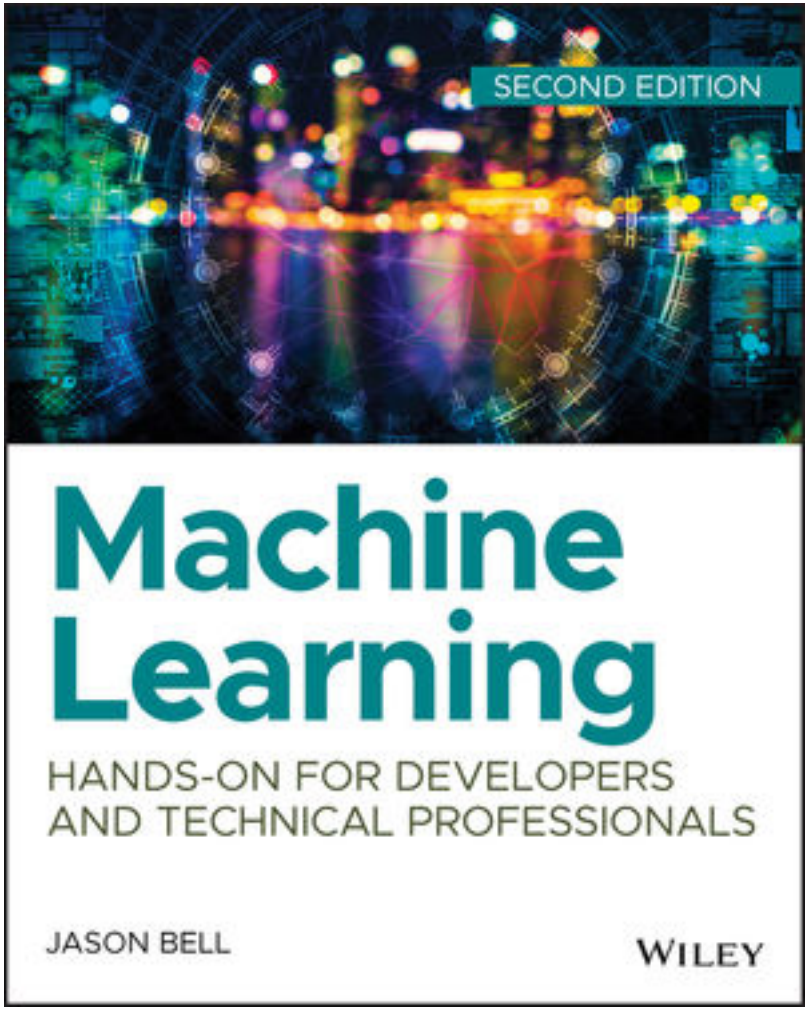
- Kafka DevOps for Digitalis
- Been using Kafka since 2013
- Wrote this >>>>>



https://datadelinquents.dev/

Heckle Permission: Active!

- My phone is on flight mode.
- My Twitter name is @jasonbelldata
- Say what you want, please add #nxnwtech #kafka hashtags as well.

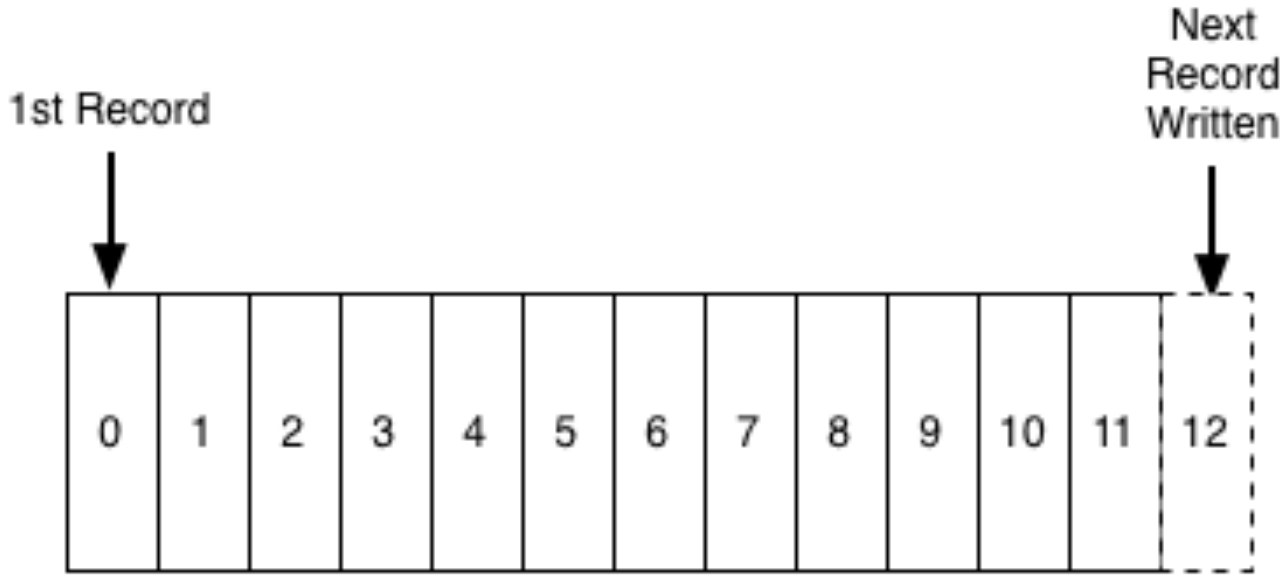


What is Kafka?

It’s an immutable log.

Thank you. Q&A?

An immutable log…..


1st Record
Next
Record
Written
0
1
2
3
4
5
6
7
8
9
10
11
12

Think of it like this…..

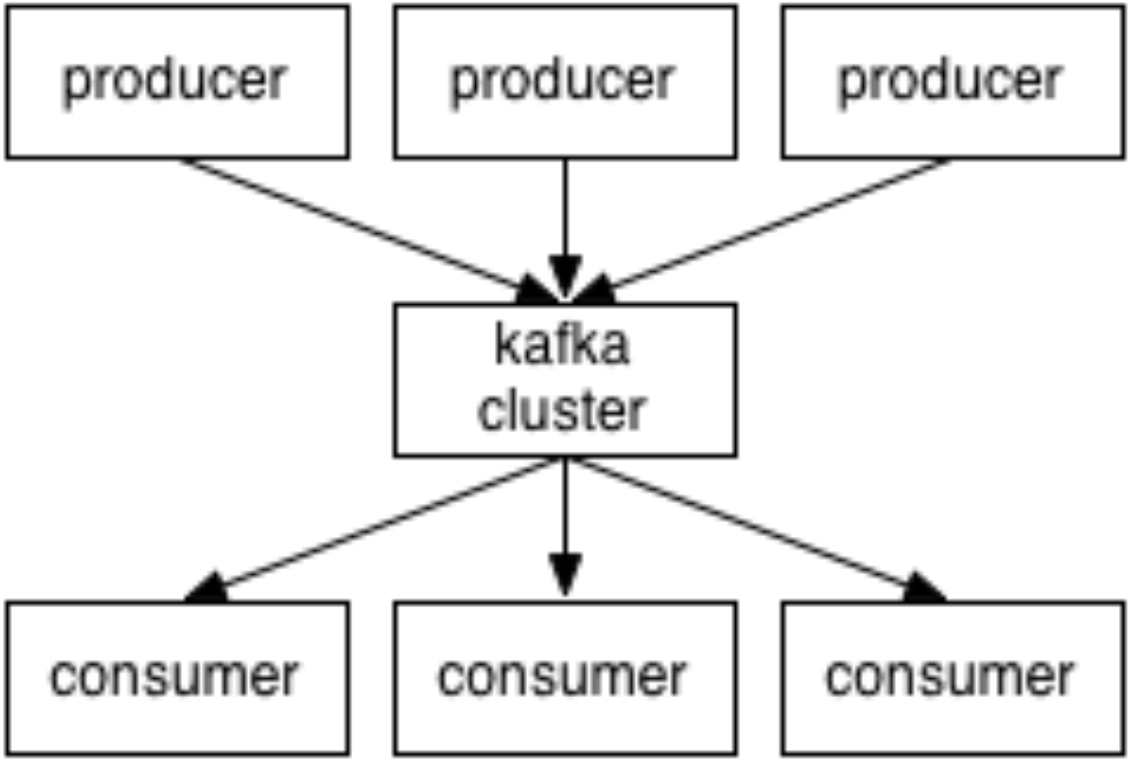

B
My Kafka Log...
Normal Bias
EQ-120μs
Noise
Reduction
IN
OUT
D-C90
TDK

So, what is Kafka *really*?

Kafka is an event processing platform which is based on a distributed architecture.

To producers and consumer subscribed to the system it would appear as a standalone processing engine but production systems are built on many machines.

It can handle millions of messages throughput, dependent on physical disk and RAM on the machines, and is fault tolerant.


producer
producer
producer
kafka
cluster
consumer
consumer
consumer

Messages are sent to topics which are written sequentially in an immutable log. Kafka can support many topics and can be replicated and partitioned.

Once the records are appended to the topic log they can’t be deleted or amended.

(If the customer wants you to
“replay from the start”, well they can’t)

It’s a very simple data structure where each message is byte encoded.

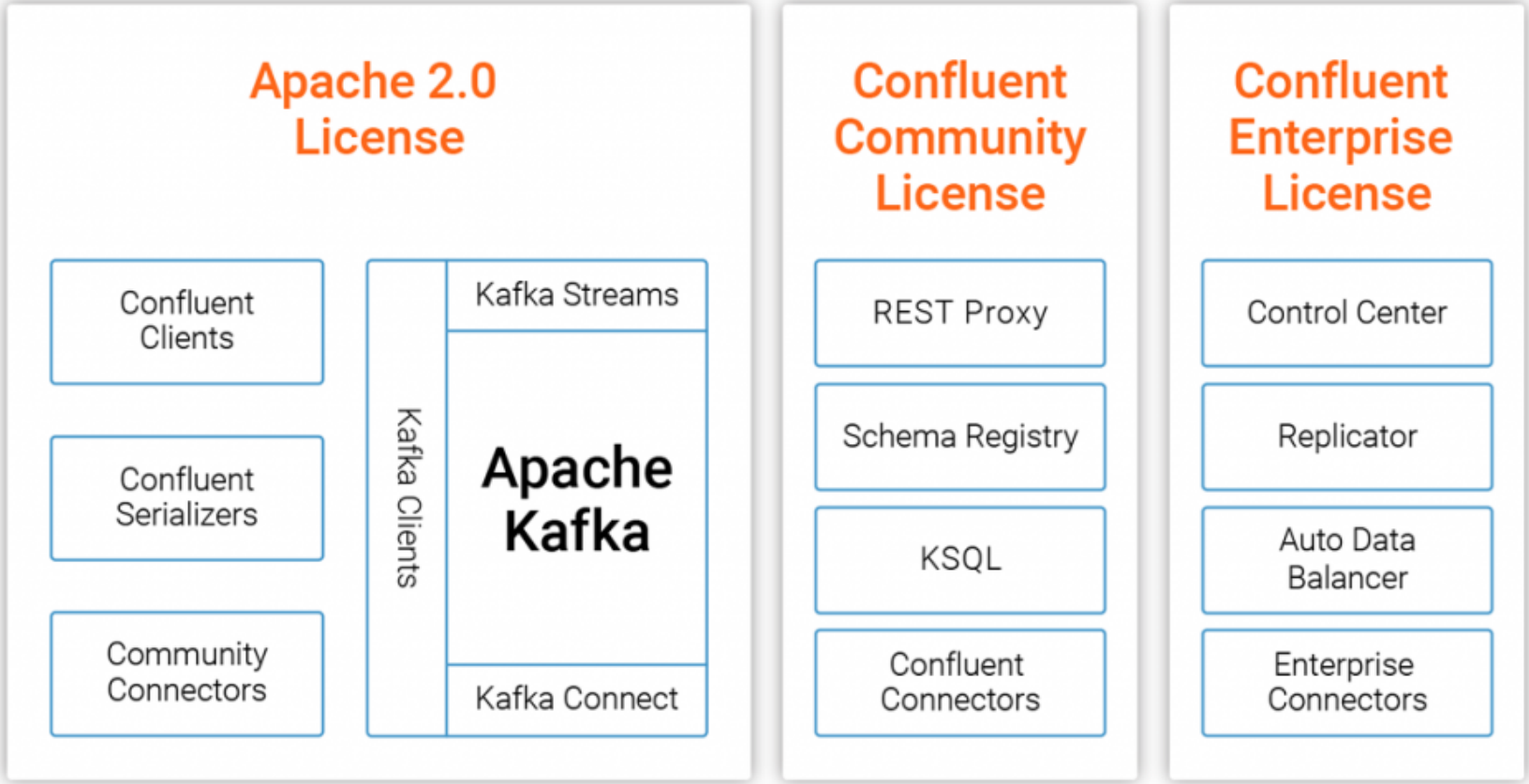
Each Message has:

A key
A header (optional)
A value (payload)
A timestamp

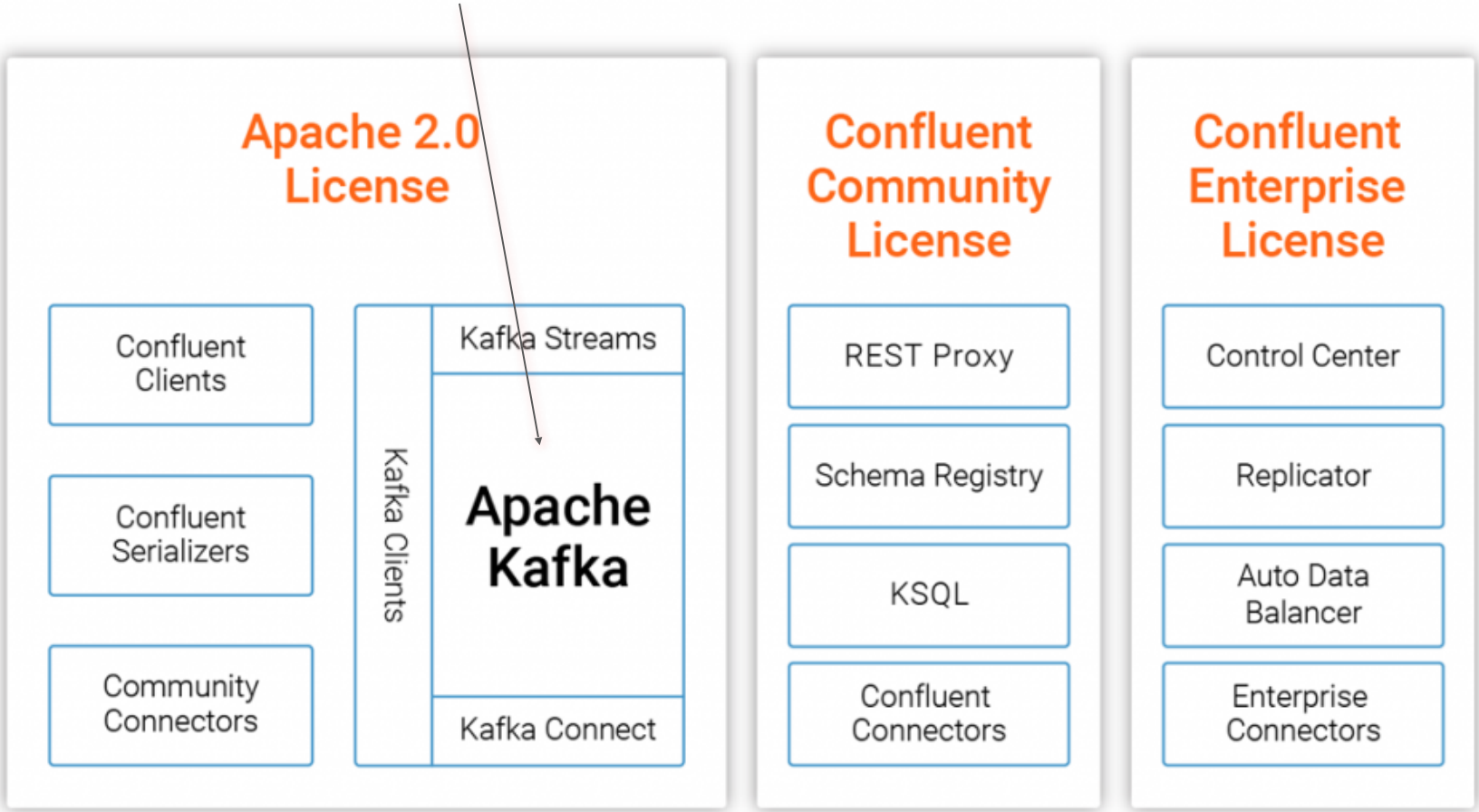
Producers and consumers can serialise and deserialise data to various formats.

(I'll talk about that later)

The Kafka Ecosystem


Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

The Kafka Cluster

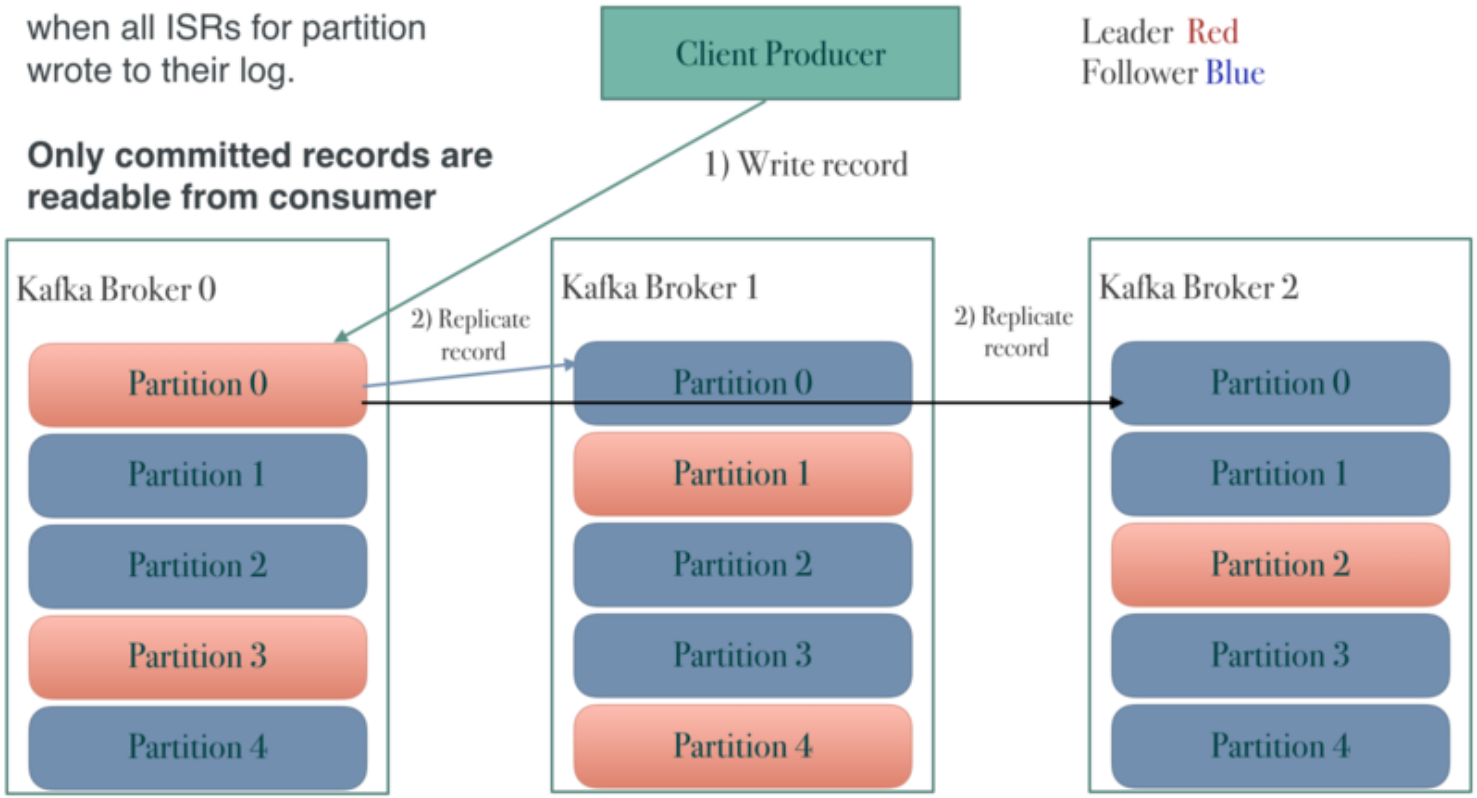

Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

Brokers

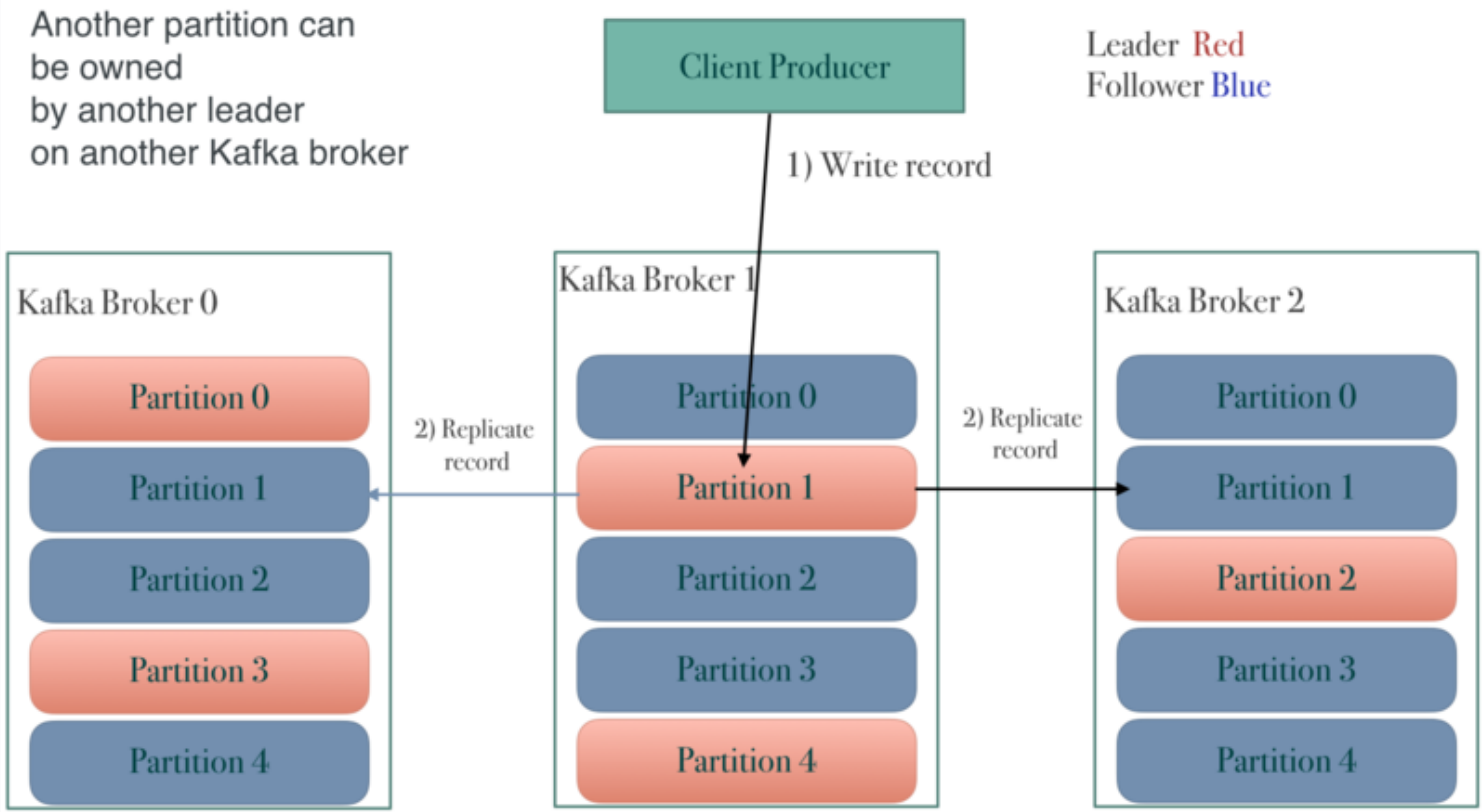
Kafka Replication to Partition 0

Record is considered "committed" when all ISRs for partition wrote to their log.

Only committed records are readable from consumer



Kafka Replication to Partitions 1



Topics

Creating a Topic

bin/kafka-topics --create –zookeeper localhost:2181 --replication-factor 4 --partitions 10 --topic testtopic --config min.insync.replicas=2

Rentention……

Rentention by Time

log.retention.hours
log.retention.minutes
log.retention.ms

The default is 168 hours (7 days) if not set.

Note: The smallest value setting take priority, be careful!

Rentention by Size

log.retention.bytes

Defines volume of log bytes retained. Applied per partition, so if set to 1GB and you have 8 partitions, you are storing 8GB.

Client Libraries

Client Library Language Support

C/C++	github.com/edenhill/librdkafka
Go	github.com/confluentinc/confluent-kafka-go
Java	Kafka Consumer and Kafka Producer
JMS	JMS Client
.NET	github.com/confluentinc/confluent-kafka-dotnet
Python	github.com/confluentinc/confluent-kafka-python

Producers

Producers
(Sends messages to the Kafka Cluster)

```
import java.util.Properties;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
public class SimpleProducer {
   public static void main(String[] args) throws Exception{
      String topicName = “testtopic”;

      Properties props = new Properties();
      props.put("bootstrap.servers", “localhost:9092");
      props.put("acks", “all");
      props.put("retries", 0);
      props.put("batch.size", 16384);
      props.put("linger.ms", 1);
      props.put("buffer.memory", 33554432);
      props.put("key.serializer",
         "org.apache.kafka.common.serialization.StringSerializer");
      props.put("value.serializer",
         "org.apache.kafka.common.serialization.StringSerializer");

      Producer<String, String> producer = new KafkaProducer
         <String, String>(props);

      for(int i = 0; i < 10; i++)
         producer.send(new ProducerRecord<String, String>(topicName,
            Integer.toString(i), Integer.toString(i)));
               System.out.println(“Message sent successfully”);
               producer.close();
   }
}
```

```
import java.util.Properties;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
public class SimpleProducer {
   public static void main(String[] args) throws Exception{
      String topicName = “testtopic”;

      Properties props = new Properties();
      props.put("bootstrap.servers", “localhost:9092");
      props.put("acks", “all");
      props.put("retries", 0);
      props.put("batch.size", 16384);
      props.put("linger.ms", 1);
      props.put("buffer.memory", 33554432);
      props.put("key.serializer",
         "org.apache.kafka.common.serialization.StringSerializer");
      props.put("value.serializer",
         "org.apache.kafka.common.serialization.StringSerializer");

      Producer<String, String> producer = new KafkaProducer
         <String, String>(props);

      for(int i = 0; i < 10; i++)
         producer.send(new ProducerRecord<String, String>(topicName,
            Integer.toString(i), Integer.toString(i)));
               System.out.println(“Message sent successfully”);
               producer.close();
   }
}
```

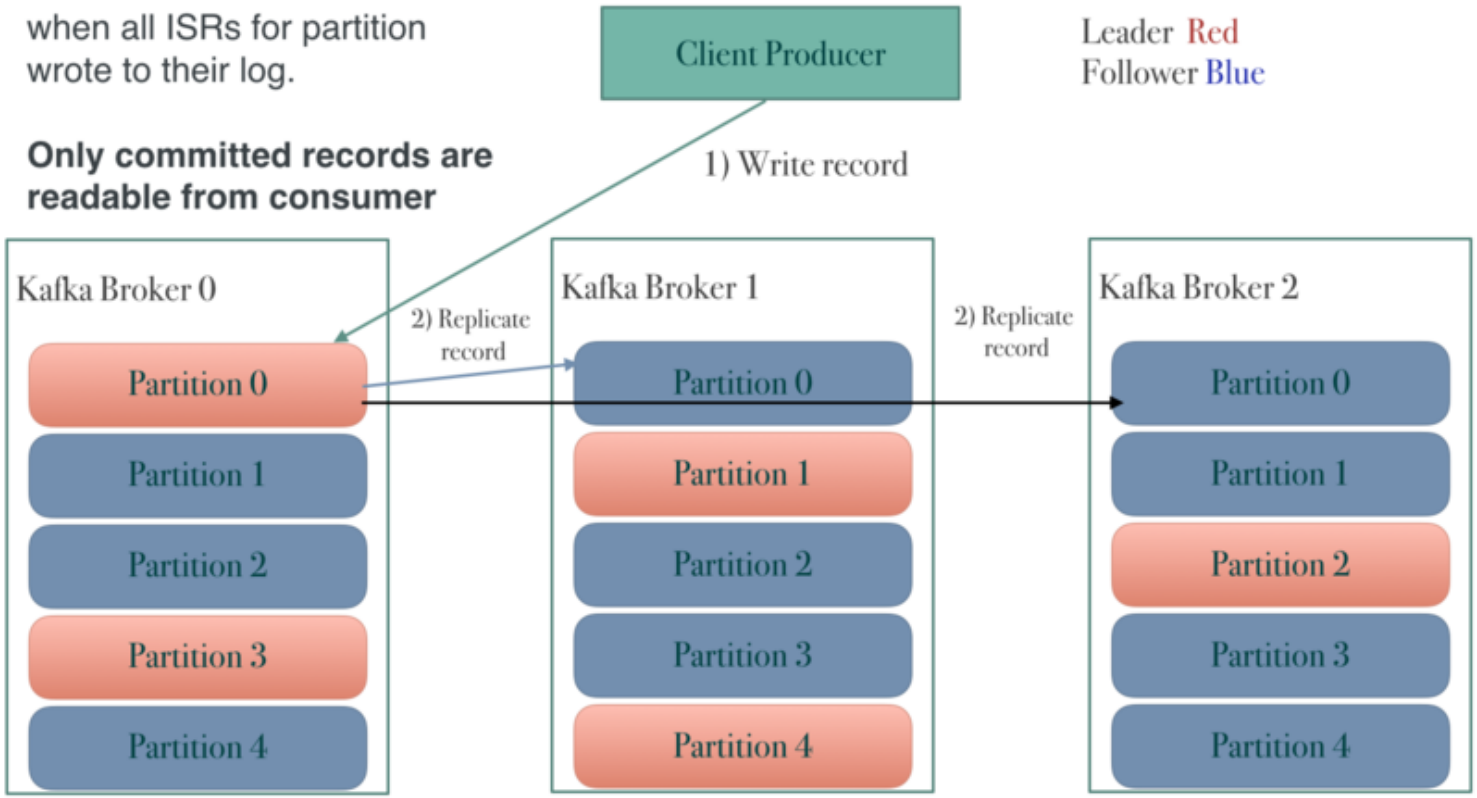
props.put("acks", “all");

Value	Action
-1/all	Message has been received by the leader and followers.
0	Fire and Forget
1	Once the message is received by the leader.

Kafka Replication to Partition 0

Record is considered "committed" when all ISRs for partition wrote to their log.

Only committed records are readable from consumer



Serialize/Deserialize Types

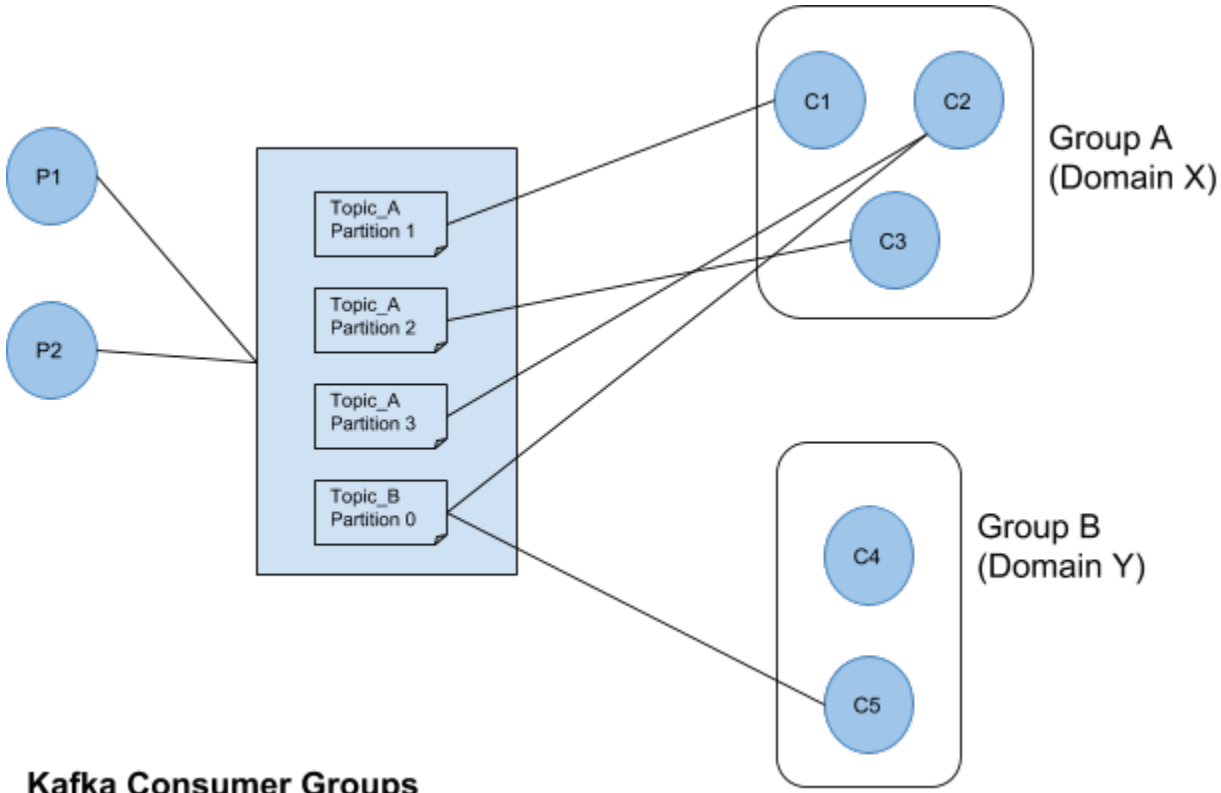
byte[]	Serdes.ByteArray(), Serdes.Bytes()
ByteBuffer	Serdes.ByteBuffer()
Double	Serdes.Double()
Integer	Serdes.Integer()
Long	Serdes.Long()
String	Serdes.String()
JSON	JsonPOJOSerializer()/Deserializer()

```
public class ProducerWithCallback implements ProducerInterceptor{

    private int onSendCount;
    private int onAckCount;
    private final Logger logger = LoggerFactory.getLogger(ProducerWithCallback.class);

    @Override
    public ProducerRecord onSend(final ProducerRecord record) {
        onSendCount++;
        System.out.println(String.format("onSend topic=%s key=%s value=%s %d \n",
                record.topic(), record.key(), record.value().toString(),
                record.partition()));

        return record;
    }

    @Override
    public void onAcknowledgement(final RecordMetadata metadata, final Exception exception) {
        onAckCount++;
        System.out.println(String.format("onAck topic=%s, part=%d, offset=%d\n",
                metadata.topic(), metadata.partition(), metadata.offset()));
    }

    @Override
    public void close() {
        System.out.println("Total sent: " + onSendCount);
        System.out.println("Total acks: " + onAckCount);
    }

    @Override
    public void configure(Map<String,?> configs) {

    }
}
```

Consumers

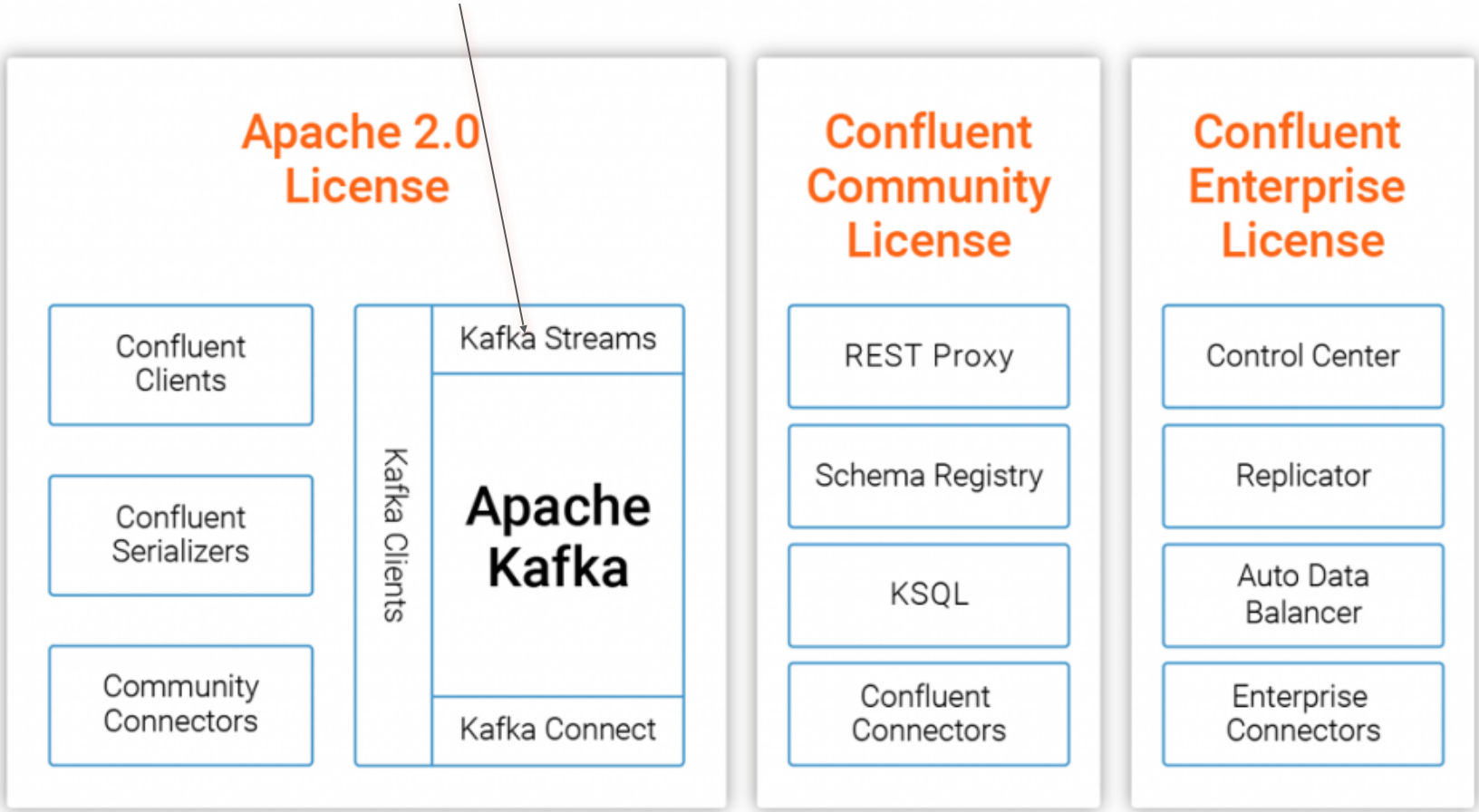
```
public class SimpleConsumer {
  public static void main(String[] args) throws Exception {
    if(args.length == 0){
      System.out.println("Enter topic name");
      return;
    }
    //Kafka consumer configuration settings
    String topicName = args[0].toString();
    Properties props = new Properties();

    props.put("bootstrap.servers", "localhost:9092");
    props.put("group.id", "test");
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "1000");
    props.put("session.timeout.ms", "30000");
    props.put("key.deserializer",
      "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer",
      "org.apache.kafka.common.serialization.StringDeserializer");
    KafkaConsumer<String, String> consumer = new KafkaConsumer
      <String, String>(props);

    //Kafka Consumer subscribes list of topics here.
    consumer.subscribe(Arrays.asList(topicName))

    //print the topic name
    System.out.println("Subscribed to topic " + topicName);
    int i = 0;

    while (true) {
      ConsumerRecords<String, String> records = con-sumer.poll(100);
      for (ConsumerRecord<String, String> record : records)

      // print the offset,key and value for the consumer records.
      System.out.printf("offset = %d, key = %s, value = %s\n",
        record.offset(), record.key(), record.value());
    }
  }
}
```

Consumer Groups



Kafka Consumer Groups

Streaming API


Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

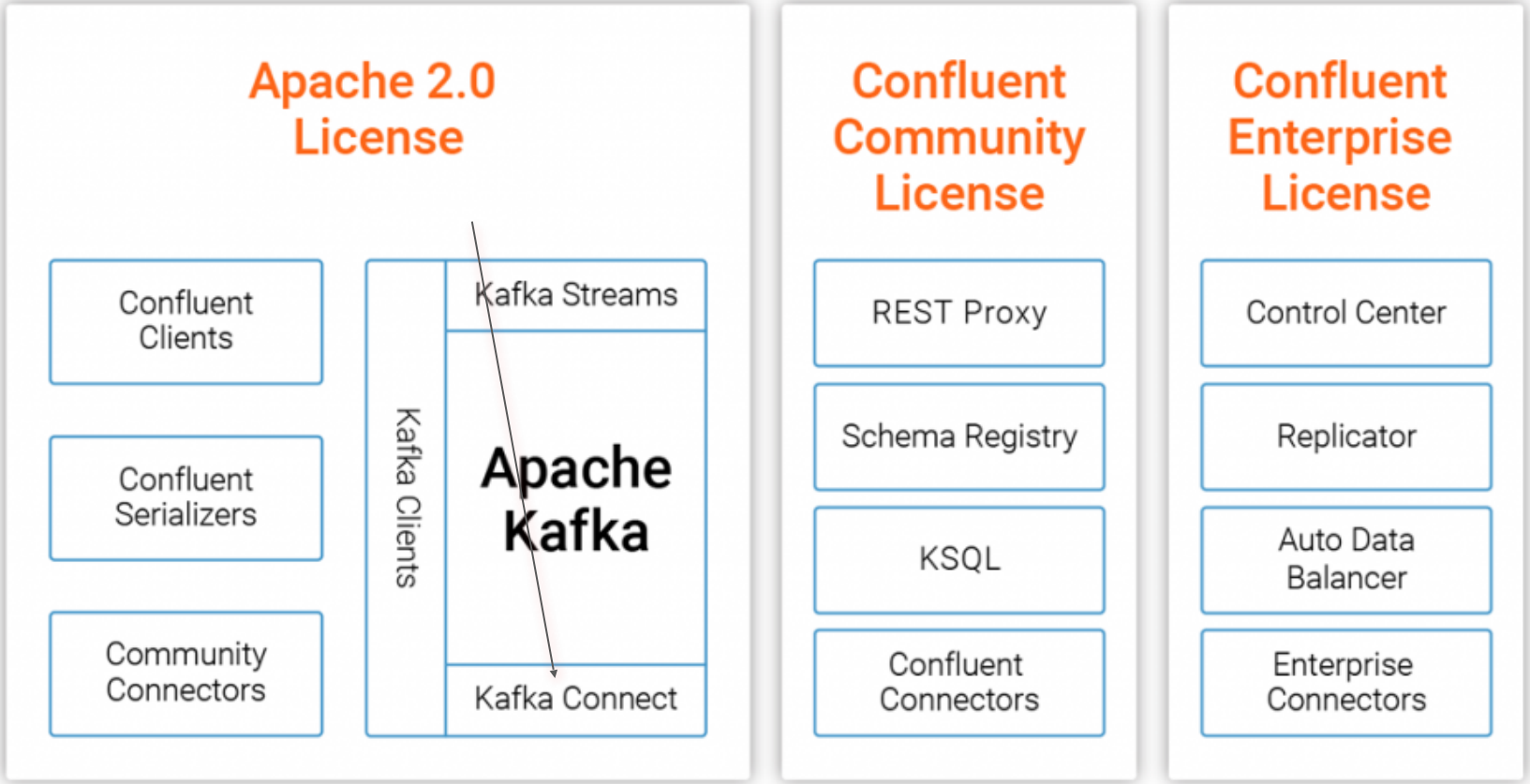
```
public class WordCountLambdaExample {

  static final String inputTopic = "streams-plaintext-input";
  static final String outputTopic = "streams-wordcount-output";

  /**
   * The Streams application as a whole can be launched like any normal Java application that has a `main()` method.
   */
  public static void main(final String[] args) {
    final String bootstrapServers = args.length > 0 ? args[0] : "localhost:9092";

    // Configure the Streams application.
    final Properties streamsConfiguration = getStreamsConfiguration(bootstrapServers);

    // Define the processing topology of the Streams application.
    final StreamsBuilder builder = new StreamsBuilder();
    createWordCountStream(builder);
    final KafkaStreams streams = new KafkaStreams(builder.build(), streamsConfiguration);
    streams.cleanUp();
    streams.start();
    Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
  }


  static Properties getStreamsConfiguration(final String bootstrapServers) {
    final Properties streamsConfiguration = new Properties();
    streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount-lambda-example");
    streamsConfiguration.put(StreamsConfig.CLIENT_ID_CONFIG, "wordcount-lambda-example-client");
    streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
    streamsConfiguration.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    streamsConfiguration.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    streamsConfiguration.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 10 * 1000);
    streamsConfiguration.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    streamsConfiguration.put(StreamsConfig.STATE_DIR_CONFIG, TestUtils.tempDirectory().getAbsolutePath());
    return streamsConfiguration;
  }

  static void createWordCountStream(final StreamsBuilder builder) {
    final KStream<String, String> textLines = builder.stream(inputTopic);
    final Pattern pattern = Pattern.compile("\\W+", Pattern.UNICODE_CHARACTER_CLASS);

    final KTable<String, Long> wordCounts = textLines
          .flatMapValues(value -> Arrays.asList(pattern.split(value.toLowerCase())))
                                      .groupBy((keyIgnored, word) -> word)
                                      .count();

    wordCounts.toStream().to(outputTopic, Produced.with(Serdes.String(), Serdes.Long()));
  }

}
```

Supports exactly once transactions! Yay!

I know what you're thinking…...

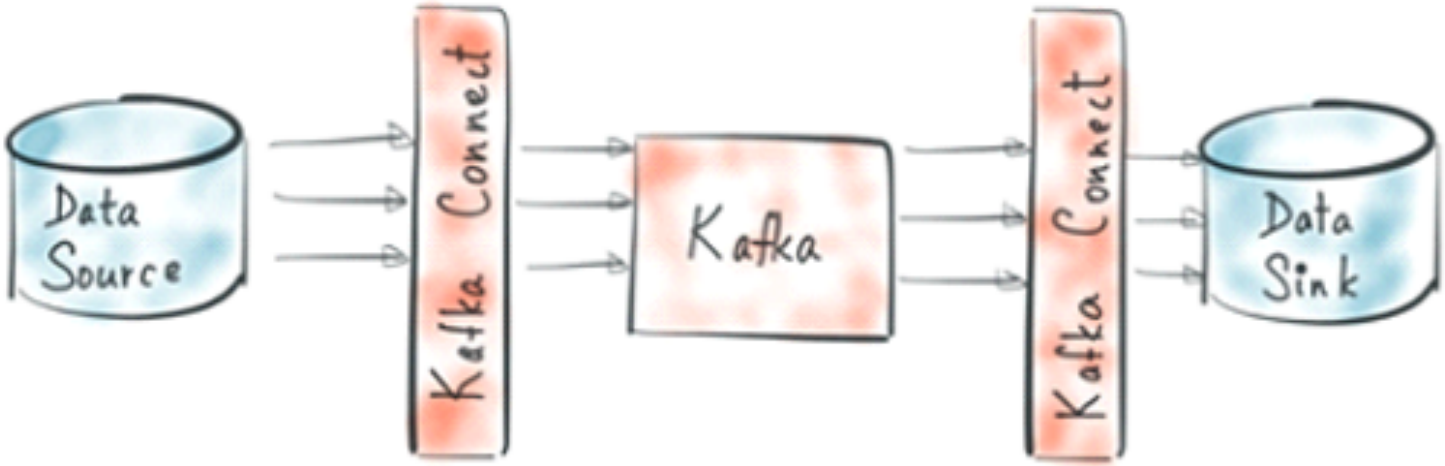
processing.guarantee=exactly_once

What is Kafka Connect?


Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

Provides a mechanism for data sources and data sinks.

KAFKA CONNECT



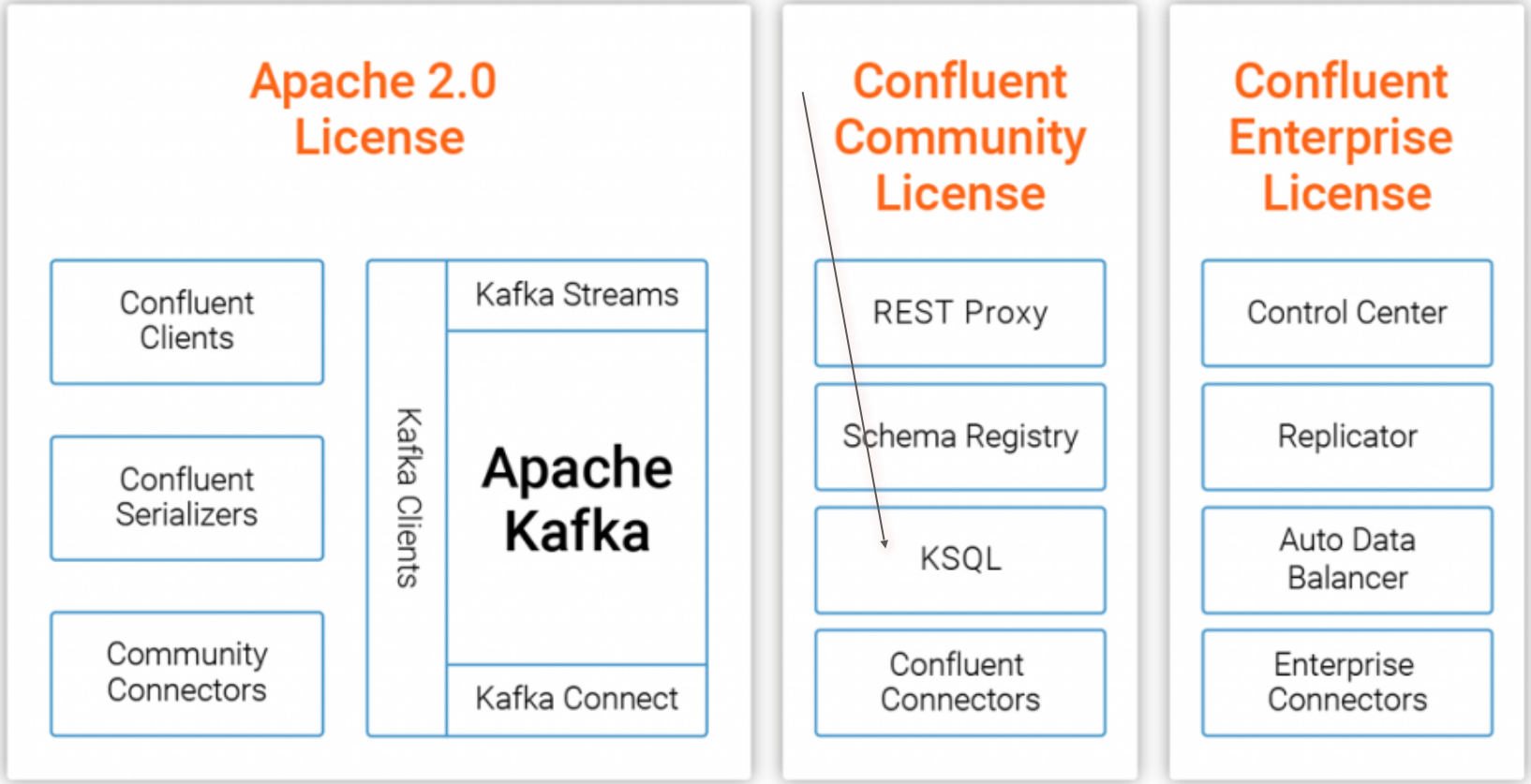
Data Source

Data from a thing (Database, Twitter stream, Logstash etc) going to Kafka.

Data Sink

Data going from Kafka to a thing (Database, file, ElasticSearch)

KSQL


Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

What is KSQL

IT

IS

NOT

A

DATABASE


amc

It’s an abstraction of the Streaming API

A very quick Kafka Connect/KSQL Concept

Pulling a Twitter Feed into KSQL

1.Get Data from Twitter Into a Kafka Topic

```
{
 "name": "twitter_source_json_01",
 "config": {
   "connector.class":
"com.github.jcustenborder.kafka.connect.twitter.TwitterSourceConnector",
   "twitter.oauth.accessToken": "",
   "twitter.oauth.consumerSecret": "",
   "twitter.oauth.consumerKey": "",
   "twitter.oauth.accessTokenSecret": "",
   "kafka.delete.topic": "twitter_deletes_json_01",
   "value.converter": "org.apache.kafka.connect.json.JsonConverter",
   "key.converter": "org.apache.kafka.connect.json.JsonConverter",
   "value.converter.schemas.enable": false,
   "key.converter.schemas.enable": false,
   "kafka.status.topic": "twitter_json_01",
   "process.deletes": true,
   "filter.keywords": "#ahashtag, #anotherhashtag"
 }
}
```

```
confluent load twitter_source -d /path/to/your/file/twitter-source.json
```

2. Create a Stream of Raw Twitter JSON Data in KSQL

```
CREATE STREAM twitter_raw (
  CreatedAt bigint,
  Id bigint,
  Text VARCHAR,
  SOURCE VARCHAR,
  Truncated VARCHAR,
  InReplyToStatusId VARCHAR,
  InReplyToUserId VARCHAR,
  InReplyToScreenName VARCHAR,
  GeoLocation VARCHAR,
  Place VARCHAR,
  Favorited VARCHAR,
  Retweeted VARCHAR,
  FavoriteCount VARCHAR,
  User VARCHAR,
  Retweet VARCHAR,
  Contributors VARCHAR,
  RetweetCount VARCHAR,
  RetweetedByMe VARCHAR,
  CurrentUserRetweetId VARCHAR,
  PossiblySensitive VARCHAR,
  Lang VARCHAR,
  WithheldInCountries VARCHAR,
  HashtagEntities VARCHAR,
  UserMentionEntities VARCHAR,
  MediaEntities VARCHAR,
  SymbolEntities VARCHAR,
  URLEntities VARCHAR)
WITH (KAFKA_TOPIC='twitter_json_01',VALUE_FORMAT='JSON');
```

3. Tranform that stream into something we actually want.

```
CREATE STREAM twitter AS \
SELECT TIMESTAMPTOSTRING(CreatedAt, 'yyyy-MM-dd HH:mm:ss.SSS') AS CreatedAt,\
EXTRACTJSONFIELD(user,'$.Name') AS user_Name,\
EXTRACTJSONFIELD(user,'$.ScreenName') AS user_ScreenName,\
EXTRACTJSONFIELD(user,'$.Location') AS user_Location,\
EXTRACTJSONFIELD(user,'$.Description') AS  user_Description,\
Text,hashtagentities,lang \
FROM twitter_raw ;
```

4. Do something funky like aggregate the tweets screen names.

```
ksql> SELECT user_screenname, \
        COUNT(*) \
        FROM twitter \
        WINDOW TUMBLING (SIZE 1 HOUR) \
        GROUP BY user_screenname \
        HAVING COUNT(*) > 1;
```

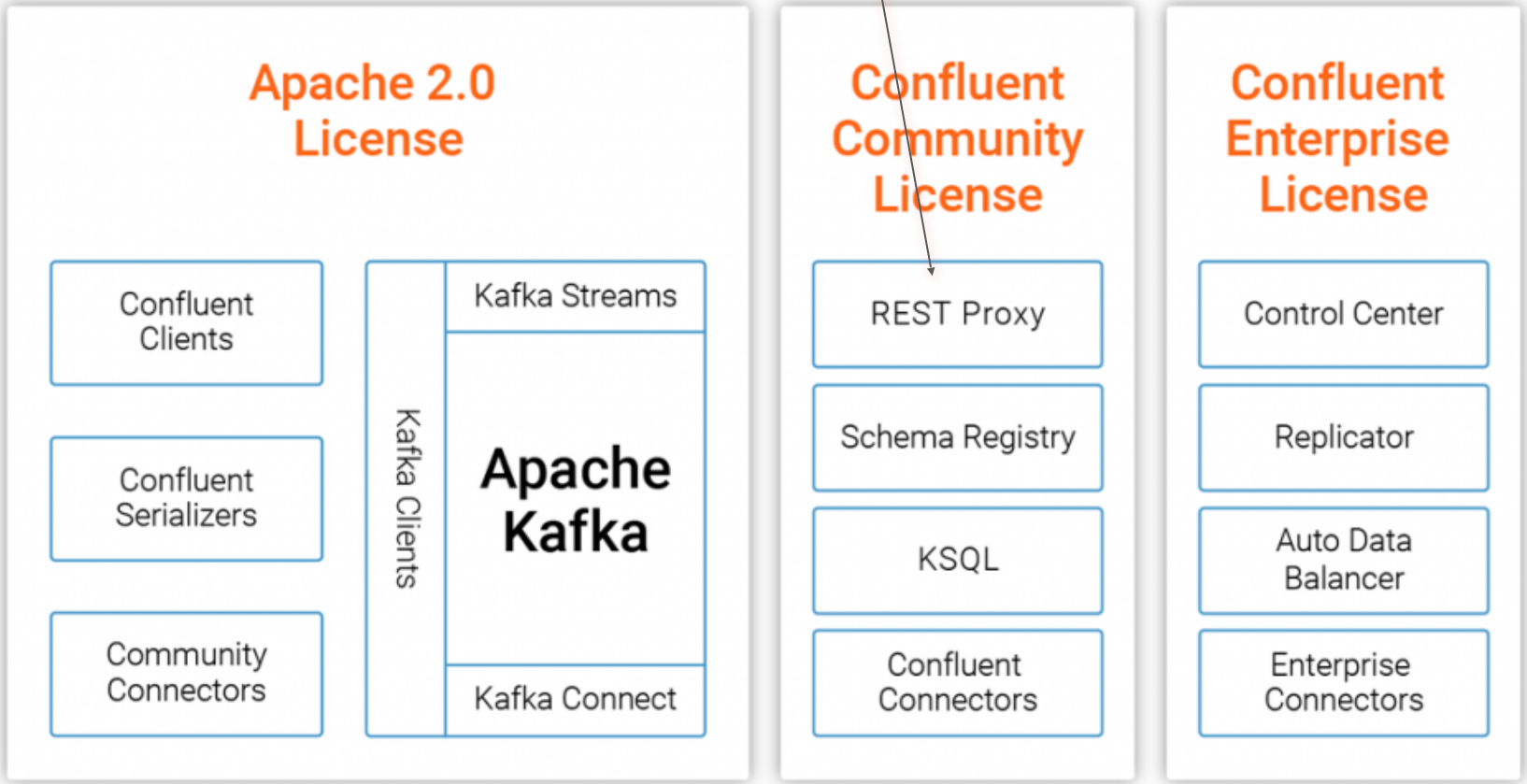
The Kafka REST API

```
# Get a list of topics
$ curl "http://localhost:8082/topics"

["__consumer_offsets","testtopic"]
```

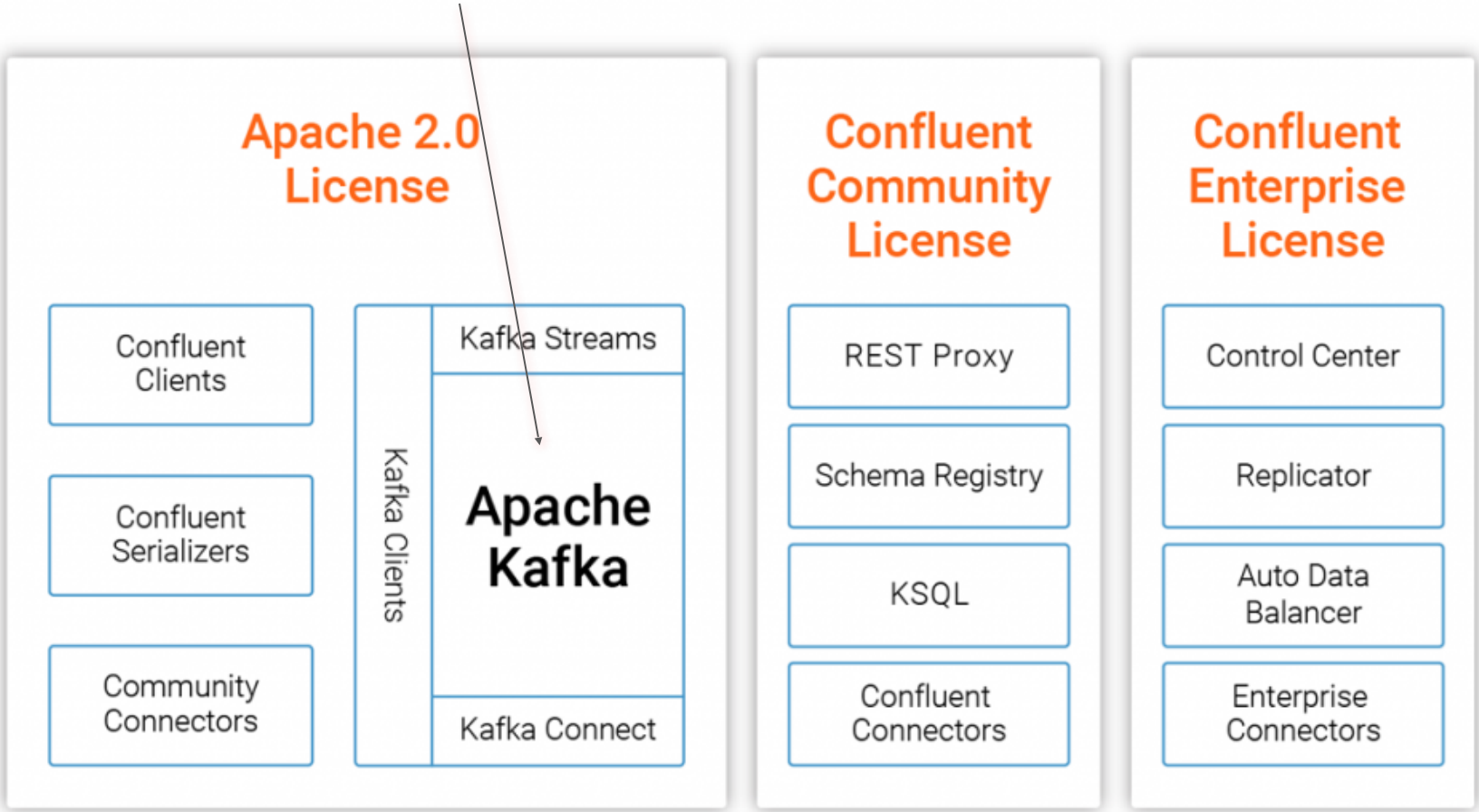
# Get info about one topic
$ curl "http://localhost:8082/topics/jsontest"

{"name":"testtopic","configs":{},"partitions":[{"partition":0,"leader":0,"replicas":[{"broker":0,"leader":true,"in_sync":true}]}]}

```
# Produce a message with JSON data
$ curl -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" \
        --data '{"records":[{"value":{"name": "testUser"}}]}' \
        "http://localhost:8082/topics/jsontest"

    {"offsets":[{"partition":0,"offset":3,"error_code":null,"error":null}],"key_schema_id":null,"value_schema_id":null}
```


Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

Schema Registry


Apache 2.0 License
Confluent Clients
Confluent Serializers
Community Connectors
Kafka Clients
Kafka Streams
Apache Kafka
Kafka Connect
Confluent Community License
REST Proxy
Schema Registry
KSQL
Confluent Connectors
Confluent Enterprise License
Control Center
Replicator
Auto Data Balancer
Enterprise Connectors

Schema Registry is a mechanism for serving and registering Avro schemas.

```
# Register a new version of a schema under the subject "Kafka-key"
$ curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" \
    --data '{"schema": "{\"type\": \"string\"}"}' \
    http://localhost:8081/subjects/Kafka-key/versions
  {"id":1}
```

```
# Register a new version of a schema under the subject "Kafka-value"
$ curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" \
    --data '{"schema": "{\"type\": \"string\"}"}' \
     http://localhost:8081/subjects/Kafka-value/versions
  {"id":1}
```

```
# List all subjects
$ curl -X GET http://localhost:8081/subjects
    ["Kafka-value","Kafka-key"]
```

```
# Delete version 3 of the schema registered under subject "Kafka-value"
$ curl -X DELETE http://localhost:8081/subjects/Kafka-value/versions/3
  3
```

```
# Delete all versions of the schema registered under subject "Kafka-value"
$ curl -X DELETE http://localhost:8081/subjects/Kafka-value
  [1, 2, 3, 4, 5]
```

Monday 2nd November at 11pm

The Cleveland Kafka Meetup

From Message to Cluster - A Realworld Introduction to Kafka Capacity Planning.

https://www.meetup.com/Cleveland-Kafka/events/274191592/

Thank you.

https://digitalis.io

http://www.itsallabet.com

@jasonbelldata