KAFKA

01 Conceito

Conceito

Plataforma de streaming em modo distribuído com as seguintes principais capacidades:

- Publicar e subscrever fluxos de registos, semelhante aos sistemas em queue ou enterprise messaging system;
- Armazenar fluxo de registos com especial cuidado à tolerância à falha;
- Processar fluxo de registos, assim que ocorrem.

Conceitos base:

- Kafka é corrido em modo cluster num ou mais servidores que podem ser divididos em vários datacenters;
- Armazena o fluxo de registos em categorias chamadas “topics”;
- Cada Registo tem uma chave, um valor e o timestamp.

02

Uso do Kafka

Uso do Kafka

- Construção em real-time de pipelines de fluxo de dados para obterem dados entre sistemas ou aplicações de modo confiável;
- Construção em real-time de aplicações de streaming que transformam ou reagem aos fluxos de dados.

- A comunicação entre os clientes e os servidores é feita de modo simples, com grande performance e com uma linguagem através de um protocolo binário sobre TCP.
- O protocolo usado é versionado e permite compatibilidade com versões anteriores;
- Apache fornece um cliente Java para Kafka, mas tem outros clientes em outras línguas.

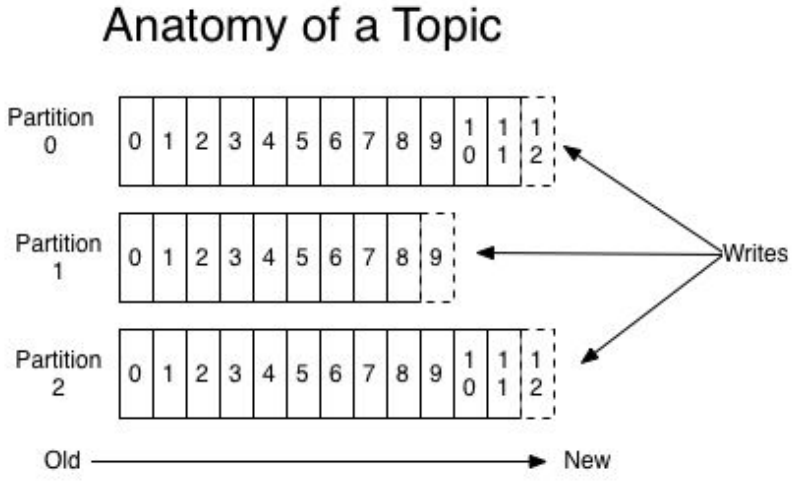
03 Topics

Topics

Os topics são sempre multi-subscriber, o topic pode ter 0,1 ou n consumidores que subscrevem os dados que nele estão escritos;

Para cada topic o Kafka reserva um log particionado como o seguinte:

- Todas as partições tem os registos ordenados e cada um dos registos tem uma identificação sequencial chamada “offset”;
- Todos os registos são persistidos pelo período de retenção de tempo configurado;
- A escolha do offset é feita por parte dos consumers.



Os registos não são apagados depois de consumidos ao contrário do sistema de queue;

- Qual a razão que não apaga o registo?

Porquê particionar os logs?

- Permite adaptar o tamanho do log ao tamanho de cada servidor;
- Serve como unidade de paralelismo.

04 Sistema Distribuído

As partições de logs são distribuídas pelos servidores do cluster;

Cada partição é replicada por um número configurado de servidores de modo a garantir tolerância à falha;

Cada partição tem pelo menos um servidor que atuará como “leader” e pode ter um ou mais servidores que agem como “followers”;

Cada servidor age como “leader” para algumas partições e como “follower” para outras de modo a haver balanceamento.

O que acontece caso o “leader” termine?

- Um dos “followers” assume a responsabilidade de “leader”;

Capacidade das mensagens serem replicadas por vários datacenters e cloud regions através do Kafka MirrorMaker.

05 Producers e Consumers

Producers e Consumers

Os producers publicam os dados nos topics que escolhem;

O producer é responsável por escolher em que partição do tópico que o registo vai ser atribuído;

Os consumers rotulam-se com um nome de grupo;

Os registos publicados num tópico são entregues aos consumers que tenham subscrito o grupo;

Os consumers podem estar em processos e/ou máquinas diferentes;

Se os consumers forem do mesmo grupo os registos serão balanceados;

Se os consumers forem de grupos diferentes os registos serão enviados para todos os consumers;

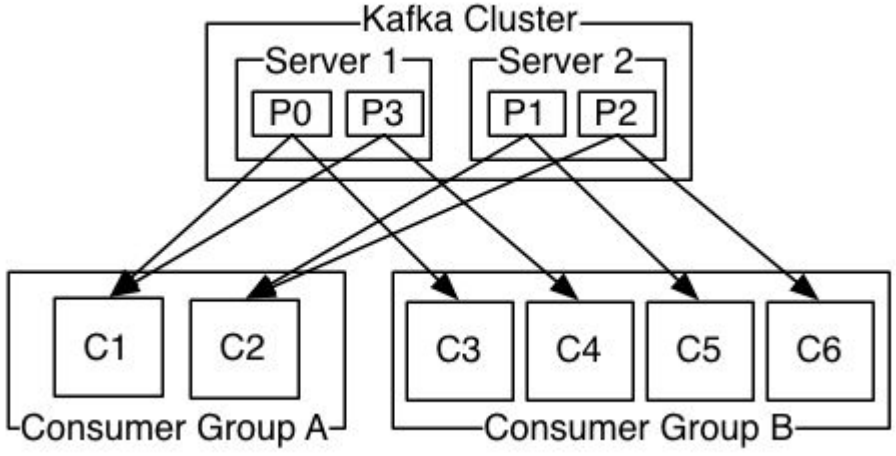
06 Message System

Message System

O conceito de grupo de consumers do Kafka generaliza o modelo queuing e publish-subscribe;

Consumer group permite dividir o processamento de uma coleção de processos;

Kafka permite fazer o broadcast de mensagens para vários grupos de consumers;



07 Streaming Processing

Streaming Processing

Para o processamento simples apenas são apenas necessárias as API’s de consumer e producer;

Para transformações mais complexas o Kafka disponibiliza a API de Streams;

08 APIS

Producer API

- Permite às aplicações enviar streams de dados para topics no Kafka Cluster
- Para usar esta api como uma dependencia de maven:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
```

- A classe principal é org.apache.kafka.clients.producer.KafkaProducer

```
Properties props = new Properties();
 props.put("bootstrap.servers", "localhost:9092");
 props.put("acks", "all");
 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<>(props);
 for (int i = 0; i < 100; i++)
     producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(i), Integer.toString(i)));

 producer.close();
```

Consumer API

- Permite às aplicações ler streams de dados dos topics no Kafka Cluster
- Para usar esta api como uma dependencia de maven:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
```

- A classe principal é org.apache.kafka.clients.consumer.KafkaConsumer

```
Properties props = new Properties();
     props.setProperty("bootstrap.servers", "localhost:9092");
     props.setProperty("group.id", "test");
     props.setProperty("enable.auto.commit", "true");
     props.setProperty("auto.commit.interval.ms", "1000");
     props.setProperty("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
     props.setProperty("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
     KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
     consumer.subscribe(Arrays.asList("foo", "bar"));
     while (true) {
         ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
         for (ConsumerRecord<String, String> record : records)
             System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());
     }
```

Streams API

- Permite transformar streams de dados de topics de entrada para topics de saída
- Para usar esta api como uma dependencia de maven:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>2.4.0</version>
</dependency>
```

- A classe principal é org.apache.kafka.clients.streams.KafkaStreams

```
Properties props = new Properties();
 props.put(StreamsConfig.APPLICATION_ID_CONFIG, "my-stream-processing-application");
 props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
 props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
 props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

 StreamsBuilder builder = new StreamsBuilder();
 builder.<String, String>stream("my-input-topic").mapValues(value ->
String.valueOf(value.length())).to("my-output-topic");

 KafkaStreams streams = new KafkaStreams(builder.build(), props);
 streams.start();
```

- Permite implementar conectores que continuamente obtêm dados de outro sistema para o Kafka ou enviam dados de outro sistema para o Kafka.

Para obter dados de outro sistema:

```
public class FileStreamSourceConnector extends SourceConnector {
    private String filename;
    private String topic;

@Override
public Class<? extends Task> taskClass() {
    return FileStreamSourceTask.class;
}

@Override
public void start(Map<String, String> props) {
    filename = props.get(FILE_CONFIG);
    topic = props.get(TOPIC_CONFIG);
}

@Override
public void stop() {
    // Não é necessário nada já que não vamos monitorizar a saida
}
```

```
 @Override
public List<Map<String, String>> taskConfigs(int maxTasks) {
    ArrayList<Map<String, String>> configs = new ArrayList<>();
    Map<String, String> config = new HashMap<>();
    if (filename != null)
        config.put(FILE_CONFIG, filename);
    config.put(TOPIC_CONFIG, topic);
    configs.add(config);
    return configs;
}
```

```
 public class FileStreamSourceTask extends SourceTask {
    String filename;
    InputStream stream;
    String topic;

    @Override
    public void start(Map<String, String> props) {
        filename = props.get(FileStreamSourceConnector.FILE_CONFIG);
        stream = openOrThrowError(filename);
        topic = props.get(FileStreamSourceConnector.TOPIC_CONFIG);
    }

    @Override
    public synchronized void stop() {
        stream.close();
    }
```

```
@Override
public List<SourceRecord> poll() throws InterruptedException {
        try {
        ArrayList<SourceRecord> records = new ArrayList<>();
        while (streamValid(stream) && records.isEmpty()) {
        LineAndOffset line = readToNextLine(stream);
        if (line != null) {
            Map<String, Object> sourcePartition = Collections.singletonMap("filename", filename);
            Map<String, Object> sourceOffset = Collections.singletonMap("position", streamOffset);
            records.add(new SourceRecord(sourcePartition, sourceOffset, topic, Schema.STRING_SCHEMA,
line));
        } else {
            Thread.sleep(1);
        }
        }
        return records;
        } catch (IOException e) {
        }
        return null;
}
```

Para enviar dados para outro sistema em vez de o conector extender de SourceConnector tem que extender de SinkConnector, sendo que a task que trata do envio dos dados em vez extender de SourceTask tem que extender de SinkTask :

```
public abstract class SinkTask implements Task {
    public void initialize(SinkTaskContext context) {
        this.context = context;
    }

    public abstract void put(Collection<SinkRecord> records);

    public void flush(Map<TopicPartition, OffsetAndMetadata> currentOffsets) {
    }
```

Admin API

- Permite gerir e inspecionar topics, brokers,acts e outros objetos do Kafka
- Para usar esta api como uma dependencia de maven:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
```

- Para criar um client de Administração é necessário implementar a interface org.apache.kafka.clients.admin.Admin

09 Hands On

Hands On

- Download https://www.apache.org/dyn/closer.cgi?path=/kafka/1.0.0/kafka_2.11-1.0.0.tgz
- Iniciar o zookeeper
    - bin/zookeeper-server-start.sh config/zookeeper.properties
- Criar 3 ficheiros um para cada nó do cluster
    - cp config/server.properties config/server.1.properties
    - cp config/server.properties config/server.2.properties
    - cp config/server.properties config/server.3.properties
- Editar os 3 ficheiros, pois existem 3 propriedades que tem que ser únicas para cada nó que são:
    - broker.id=0
    - listeners=PLAINTEXT://:9092
    - log.dirs=/tmp/kafka-logs
- Criar as pastas de logs referenciadas em cada ficheiro anteriormente editado
- Iniciar os servidores
    - bin/kafka-server-start.sh config/server.1.properties
    - bin/kafka-server-start.sh config/server.2.properties
    - bin/kafka-server-start.sh config/server.3.properties

Hands On

- Criar um topic no zookeeper
    - bin/kafka-topics.sh --create --topic my-kafka-topic --zookeeper localhost:2181 --partitions 3 --replication-factor 2
- Listar topic no zookeeper
    - bin/kafka-topics.sh --list --zookeeper localhost:2181
- Ver as partições do Topic
    - bin/kafka-topics.sh --describe --zookeeper localhost:2181 -topic my-kafka-topic
- Criar um producer para publicar no topic
    - bin/kafka-console-producer.sh --broker-list localhost:9093,localhost:9094,localhost:9095 --topic my-kafka-topic
- Criar um consumer para subscrever o topic
    - bin/kafka-console-consumer.sh --bootstrap-server localhost:9093 --topic my-kafka-topic --from-beginning

O que acontece se desligarmos o nó1??????

Hands On

- Criar aplicações através do maven:
    - mvn archetype:generate -DarchetypeGroupId=org.apache.kafka -DarchetypeArtifactId=streams-quickstart-java -DarchetypeVersion=2.4.0 -DgroupId=streams.examples -DartifactId=streams.examples -Dversion=0.1 -Dpackage=myapps
- mvn clean package
- mvn exec:java -Dexec.mainClass=streams.examples.LineSplit

VAMOS AGORA FAZER ALTERAÇÕES PARA TESTAR A STREAM CRIADA

Hands On

- Iniciar o Kafka connect
  - bin/connect-distributed.sh config/connect-distributed.properties

- Adicionar um connector através do terminal:
  - curl -s -X POST -H "Content-Type: application/json" --data '{"name": "local-file-source","config": {"connector.class": "FileStreamSource","tasks.max": "1","file": "/home/pmatrola/test","topic": "my-kafka-topic","name": "local-file-source"}}' http://localhost:8083/connectors

Podem encontrar mais informação em:

https://kafka.apache.org/24/documentation/

Linkare

FACING INNOVATION TOGETHER

PORTUGAL
+351 213 590 623
www.linkare.pt

BELGIUM
+32 280 842 60
www.linkare.be

info@linkare.com
sales@linkare.com

Pedro Matrola
Tech Leader

+351 910 527 219

pmatrola@linkare.com