Lab 6.1: Kafka Advanced Consumer Part 1

Welcome to the session 6 lab 1. The work for this lab is done in `~/kafka-training/lab6.1`.

In this lab, you are going to set up an advanced Kafka Consumer.

Please refer to the Kafka course notes for any updates or changes to this lab.

Find the latest version of this lab here.

Kafka Consumers

A consumer is a type of Kafka client that consumes records from Kafka cluster. The Kafka *Consumer* automatically handles Kafka broker failure, adapt as topic partitions leadership moves in Kafka cluster. The consumer works with Kafka broker to form consumer groups and load balance consumers. The consumer maintains connections to Kafka brokers in the cluster. The consumer must be closed to not leak resources. The Kafka client API for Consumers are ***NOT*** thread-safe.

Lab Creating an Advanced Kafka Consumer

Stock Price Consumer

The Stock Price Consumer example has the following classes:

- `StockPrice` - holds a stock price has a name, dollar, and cents
- `SimpleStockPriceConsumer` - consumes StockPrices and display batch lengths for the poll method call
- `StockAppConstants` - holds topic and broker list
- `StockPriceDeserializer` - can deserialize a ***StockPrice*** from ***byte[]***

StockPriceDeserializer

The `StockPriceDeserializer` calls the JSON parser to parse JSON in bytes to a `StockPrice` object.

~/kafka-training/lab6.1/src/main/java/com/cloudurable/kafka/consumer/StockPriceDeserializer.java

Kafka Consumer: StockPriceDeserializer - Parse JSON in bytes to a StockPrice object

```
package com.cloudurable.kafka.consumer;

import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.common.serialization.Deserializer;

import java.nio.charset.StandardCharsets;
import java.util.Map;

public class StockDeserializer implements Deserializer<StockPrice> {

    @Override
    public StockPrice deserialize(final String topic, final byte[] data) {
        return new StockPrice(new String(data, StandardCharsets.UTF_8));
    }

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

    @Override
    public void close() {
    }
}
```

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/StockPriceDeserializer.java` and follow the instructions in the file.

~/kafka-training/lab6.1/src/main/java/com/cloudurable/kafka/model/StockPrice.java

Kafka Producer: StockPrice

```
package com.cloudurable.kafka.producer.model;

import io.advantageous.boon.json.JsonFactory;

public class StockPrice {

    private final int dollars;
    private final int cents;
    private final String name;

    public StockPrice(final String json) {
        this(JsonFactory.fromJson(json, StockPrice.class));
    }
    . . .
}
```

Fix the constructor by using the hint which calls the JSON parser.

ACTION - EDIT `src/main/java/com/cloudurable/kafka/model/StockPrice.java` and follow the instructions in the file.

SimpleStockPriceKafkaConsumer

`SimpleStockPriceKafkaConsumer` has a `createConsumer` method to create a KafkaProducer instance, subscribes to stock-prices topics and has a custom deserializer. It has a `runConsumer()` method that drains topic, creates map of current stocks and calls `displayRecordsStatsAndStocks()` method. The method `displayRecordsStatsAndStocks()` prints out the size of each partition read and total record count and prints out each stock at its current price.

~/kafka-training/lab6.1/src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java

Kafka Consumer: SimpleStockPriceConsumer -

```
package com.cloudurable.kafka.consumer;
import com.cloudurable.kafka.StockAppConstants;
import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.serialization.StringDeserializer;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

public class SimpleStockPriceConsumer {

    private static Consumer<String, StockPrice> createConsumer() {
        final Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
                StockAppConstants.BOOTSTRAP_SERVERS);
        props.put(ConsumerConfig.GROUP_ID_CONFIG,
                "KafkaExampleConsumer");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                StringDeserializer.class.getName());
        //Custom Deserializer
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                StockDeserializer.class.getName());
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 500);
        // Create the consumer using props.
        final Consumer<String, StockPrice> consumer =
                new KafkaConsumer<>(props);
        //Subscribe to the topic.
        consumer.subscribe(Collections.singletonList(
                StockAppConstants.TOPIC));
        return consumer;
    }
    static void runConsumer() throws InterruptedException {
```

```
        final Consumer<String, StockPrice> consumer = createConsumer();
        final Map<String, StockPrice> map = new HashMap<>();
        try {
            final int giveUp = 1000; int noRecordsCount = 0;
            int readCount = 0;
            while (true) {
                final ConsumerRecords<String, StockPrice> consumerRecords =
                        consumer.poll( timeout: 1000);
                if (consumerRecords.count() == 0) {
                    noRecordsCount++;
                    if (noRecordsCount > giveUp) break;
                    else continue;
                }
                readCount++;
                consumerRecords.forEach(record -> {
                    map.put(record.key(), record.value());
                });
                if (readCount % 100 == 0) {
                    displayRecordsStatsAndStocks(map, consumerRecords);
                }
                consumer.commitAsync();
            }
        }
        finally {
            consumer.close();
        }
        System.out.println("DONE");
    }
    private static void displayRecordsStatsAndStocks(
            final Map<String, StockPrice> stockPriceMap,
            final ConsumerRecords<String, StockPrice> consumerRecords) {
        System.out.printf("New ConsumerRecords par count %d count %d\n",
                consumerRecords.partitions().size(),
                consumerRecords.count());
        stockPriceMap.forEach((s, stockPrice) ->
                System.out.printf("ticker %s price %d.%d \n",
                        stockPrice.getName(),
                        stockPrice.getDollars(),
                        stockPrice.getCents()));
        System.out.println();
    }
    public static void main(String... args) throws Exception {
        runConsumer();
    }
}
```

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java` and follow the instructions in the file.

Running the example

To run the example, you need to run ZooKeeper, then run the three Kafka Brokers. Once that is running, you will need to run create-topic.sh. And lastly, run the `SimpleStockPriceConsumer` from the IDE.

First, run ZooKeeper.

Running ZooKeeper with run-zookeeper.sh (Run in a new terminal)

```
~/kafka-training

$ cat run-zookeeper.sh
#!/usr/bin/env bash
cd ~/kafka-training
```

```
kafka/bin/zookeeper-server-start.sh \
   kafka/config/zookeeper.properties

$ ./run-zookeeper.sh
```

Now run the first Kafka Broker.

Running the 1st Kafka Broker (Run in a new terminal)

```
~/kafka-training/lab6.1

$ cat bin/start-1st-server.sh
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training
## Run Kafka
kafka/bin/kafka-server-start.sh \
    "$CONFIG/server-0.properties"

$ bin/start-1st-server.sh
```

Now run the second Kafka Broker.

Running the 2nd Kafka Broker (Run in a new terminal)

```
~/kafka-training/lab6.1

$ cat bin/start-2nd-server.sh
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training
## Run Kafka
kafka/bin/kafka-server-start.sh \
    "$CONFIG/server-1.properties"

$ bin/start-2nd-server.sh
```

Now run the third Kafka Broker.

Running the 3rd Kafka Broker (Run in a new terminal)

```
~/kafka-training/lab6.1

$ cat bin/start-3rd-server.sh
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training
## Run Kafka
kafka/bin/kafka-server-start.sh \
    "$CONFIG/server-2.properties"

$ bin/start-3rd-server.sh
```

Once all brokers are running, run create-topic.sh as follows.

Running create topic

```
~/kafka-training/lab6.1

$ cat bin/create-topic.sh
#!/usr/bin/env bash

cd ~/kafka-training

kafka/bin/kafka-topics.sh \
    --create \
    --zookeeper localhost:2181 \
    --replication-factor 3 \
```

```
    --partitions 3 \
    --topic stock-prices \
    --config min.insync.replicas=2

$ bin/create-topic.sh
    Created topic "stock-prices".
```

The create-topics script creates a topic. The name of the topic is stock-prices. The topic has three partitions. The created topic has a replication factor of three.

For the config, only the broker id and log directory changes.

config/server-0.properties

```
broker.id=0
listeners=PLAINTEXT://localhost:9092
log.dirs=./logs/kafka-0
...
```

Run the `StockPriceKafkaProducer` from your IDE. You should see log messages from StockSender(s) with StockPrice name, JSON value, partition, offset, and time.

Run the `SimpleStockPriceConsumer` from your IDE. You should see the size of each partition read, the total record count and each stock at its current price.