Lab 6.2: StockPriceConsumer Controlling Consumer Position

Welcome to the session 6 lab 2. The work for this lab is done in `~/kafka-training/labs/lab6.2`. In this lab, you are going to control consumer position.

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

Find the latest version of this lab here.

Lab Control Consumer Position

Offsets and Consumer Position

Consumer position is the topic, partition and offset of the last record per partition consuming. Offset for each record in a partition as a unique identifier record location in the partition. Consumer position gives offset of next *(highest)* record that it consumes and the position advances automatically for each call to `poll(..)`.

Consumer committed Position

Consumer committed position is the last offset that has been stored to the Kafka broker if the consumer fails, this allows the consumer to picks up at the last committed position. Consumer can auto commit offsets ***(enable.auto.commit)*** periodically ***(auto.commit.interval.ms)*** or do commit explicitly using `commitSync()` and `commitAsync()`.

Consumer Groups

Kafka organizes Consumers into consumer groups. Consumer instances that have the same ***group.id*** are in the same consumer group. Pools of consumers in a group divide work of consuming and processing records. In the consumer groups, processes and threads can run on the same box or run distributed for ***scalability/fault tolerance***.

Kafka shares ***topic partitions*** among all consumers in a consumer group, each partition is assigned to exactly one consumer in a consumer group. ***E.g.*** One topic has six partitions, and a consumer group has two consumer process, each process gets consume three partitions. If a consumer fails, Kafka reassigned partitions from failed consumer to other consumers in the same consumer group. If new consumer joins, Kafka moves partitions from existing consumers to the new consumer. A ***Consumer group*** forms a ***single logical subscriber*** made up of multiple consumers. Kafka is a multi-subscriber system, Kafka supports *N* number of ***consumer groups*** for a given topic without duplicating data.

Partition Reassignment

Consumer partition reassignment in a consumer group happens automatically. Consumers are notified via ***ConsumerRebalanceListener*** and triggers consumers to finish necessary clean up. A Consumer can use the API to assign specific partitions using the ***assign***(Collection) method, but using `assign` disables dynamic partition assignment and consumer group coordination. Dead consumers may see `CommitFailedException` thrown from a call to `commitSync()`. Only active members of consumer group can commit offsets.

Controlling Consumers Position

You can control consumer position moving to forward or backward. Consumers can re-consume older records or skip to the most recent records.

~/kafka-training/labs/lab6.2/src/main/java/com/cloudurable/kafka/consumer/SeekTo.java

Kafka Consumer: SeekTo

```
package com.cloudurable.kafka.consumer;
public enum SeekTo {
    START, END, LOCATION, NONE
}
```

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

Use ***consumer.seek***(TopicPartition, long) to specify. ***E.g.*** `consumer.seekToBeginning(Collecion)` and `consumer.seekToEnd(Collection)` ***Use Case Time-sensitive record processing:*** Skip to most recent records. ***Use Case Bug Fix:*** Reset position before bug fix and replay log from there. ***Use Case Restore State for Restart or Recovery:*** Consumer initialize position on start-up to whatever is contained in local store and replay missed parts (cache warm-up or replacement in case of failure assumes Kafka retains sufficient history or you are using log compaction).

Managing Offsets

For the consumer to manage its own offset you just need to do the following: Set ***enable.auto.commit*** = ***false*** Use offset provided with each ConsumerRecord to save your position (partition/offset) On restart restore consumer position using ***kafkaConsumer.seek(TopicPartition, long)*** Usage like this simplest when the partition assignment is also done manually using ***assign()*** instead of ***subscribe()***. If using automatic partition assignment, you must handle cases where partition assignments changes. Pass ***ConsumerRebalanceListener*** instance in call to ***kafkaConsumer.subscribe***(Collection, ConsumerRebalanceListener) and ***kafkaConsumer.subscribe***(Pattern, ConsumerRebalanceListener). When partitions taken from consumer, commit its offset for partitions by implementing ***ConsumerRebalanceListener.onPartitionsRevoked***(Collection) and when partitions are assigned to consumer, look up offset for new partitions and correctly initialize consumer to that position by implementing ***ConsumerRebalanceListener.onPartitionsAssigned***(Collection).

~/kafka-training/labs/lab6.2/src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java

Kafka Consumer: SimpleStockPriceConsumer.Consumer

```
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 SeekTo seekTo,
                                    final long location) {
        final Properties props = initProperties();

        // Create the consumer using props.
        final Consumer<String, StockPrice> consumer =
                new KafkaConsumer<>(props);

        final ConsumerRebalanceListener consumerRebalanceListener = null;

        consumer.subscribe(Collections.singletonList(
                StockAppConstants.TOPIC), consumerRebalanceListener);
        return consumer;
    }
...
}
```

~/kafka-training/labs/lab6.2/src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java

Kafka Consumer: SimpleStockPriceConsumer.Consumer

```
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 {
...
    public static void main(String... args) throws Exception {

        SeekTo seekTo = SeekTo.NONE; // SeekTo what?
        long location = -1; // Location to seek to if SeekTo.Location
        int readCountStatusUpdate = 100;
        if (args.length >= 1) {
            seekTo = SeekTo.valueOf(args[0].toUpperCase());
            if (seekTo.equals(SeekTo.LOCATION)) {
                location = Long.parseLong(args[1]);
            }
        }
        if (args.length == 3) {
            readCountStatusUpdate = Integer.parseInt(args[2]);
        }
        runConsumer(seekTo, location, readCountStatusUpdate);
    }
...
}
```

ACTION - EDIT

`src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java` **and follow the instructions for the main method.**

Controling Consumers Position Example

~/kafka-training/labs/lab6.2/src/main/java/com/cloudurable/kafka/consumer/SeekToConsumerRebalanceListener.java

Kafka Consumer: SeekToConsumerRebalanceListener

```
package com.cloudurable.kafka.consumer;
import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.common.TopicPartition;
import java.util.Collection;
public class SeekToConsumerRebalanceListener implements ConsumerRebalanceListener {
    private final Consumer<String, StockPrice> consumer;
    private final SeekTo seekTo; private boolean done;
    private final long location;
    private final long startTime = System.currentTimeMillis();
    public SeekToConsumerRebalanceListener(final Consumer<String, StockPrice> consumer, final SeekTo seekTo,
                                           final long location) {
        this.seekTo = seekTo;
        this.location = location;
        this.consumer = consumer;
    }
    @Override
    public void onPartitionsAssigned(final Collection<TopicPartition> partitions) {
        if (done) return;
        else if (System.currentTimeMillis() - startTime > 30_000) {
            done = true;
            return;
        }
        switch (seekTo) {
            case END:                   //Seek to end
                consumer.seekToEnd(partitions);
                break;
            case START:                 //Seek to start
                consumer.seekToBeginning(partitions);
                break;
            case LOCATION:              //Seek to a given location
                partitions.forEach(topicPartition ->
                        consumer.seek(topicPartition, location));
                break;
        }
    }
}
```

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

~/kafka-training/labs/lab6.2/src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java

Kafka Consumer: SimpleStockPriceConsumer.runConsumer

```
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 void runConsumer(final SeekTo seekTo, final long location,
                                    final int readCountStatusUpdate) throws InterruptedException {
        final Map<String, StockPrice> map = new HashMap<>();
        try (final Consumer<String, StockPrice> consumer =
                     createConsumer(seekTo, location)) {
            final int giveUp = 1000; int noRecordsCount = 0; int readCount = 0;
            while (true) {
                final ConsumerRecords<String, StockPrice> consumerRecords =
                        consumer.poll(1000);
```

```
                if (consumerRecords.count() == 0) {
                    noRecordsCount++;
                    if (noRecordsCount > giveUp) break;
                    else continue;
                }
                readCount++;
                consumerRecords.forEach(record -> {
                    map.put(record.key(), record.value());
                });
                if (readCount % readCountStatusUpdate == 0) {
                    displayRecordsStatsAndStocks(map, consumerRecords);
                }
                consumer.commitAsync();
            }
        }
        System.out.println("DONE");
    }
...
}
```

Use SeekTo, Position and ReadCountStatusUpdate

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java` and follow the instructions for the runConsumer method.

~/kafka-training/labs/lab6.2/src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java

Kafka Consumer: SimpleStockPriceConsumer.Consumer

```
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 SeekTo seekTo,
                                                                final long location) {
        final Properties props = initProperties();

        // Create the consumer using props.
        final Consumer<String, StockPrice> consumer =
                new KafkaConsumer<>(props);

        // Create SeekToConsumerRebalanceListener and assign it to consumerRebalanceListener
        final ConsumerRebalanceListener consumerRebalanceListener =
                new SeekToConsumerRebalanceListener(consumer, seekTo, location);

        // Subscribe to the topic.
        consumer.subscribe(Collections.singletonList(
                StockAppConstants.TOPIC), consumerRebalanceListener);
        return consumer;
    }
...
}
```

Create ***SeekToConsumerRebalanceListener*** passing `SeekTo` and `location`

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java` and follow the instructions for the createConsumer method.

ACTION - RUN ZooKeeper and Brokers if needed.

ACTION - RUN SimpleStockPriceConsumer run with moving to start of log

ACTION - RUN StockPriceKafkaProducer

ACTION - RUN SimpleStockPriceConsumer run with moving to end of log

ACTION - RUN SimpleStockPriceConsumer run with moving to a certain location in log

It should all run. Stop consumer and producer when finished.

Auto offset reset config

What happens when you seek to an invalid location is controlled by consumer-based configuration property:

```
ConsumerConfig.AUTO_OFFSET_RESET_CONFIG = "auto.offset.reset"

Valid values: "latest", "earliest", "none"
```

- latest: automatically reset the offset to the latest offset
- earliest: automatically reset the offset to the earliest offset
- none: throw an exception to the consumer if no previous offset is found for the consumer's group anything else: throw exception to the consumer.