Lab 6.4: StockPriceConsumer Exactly Once Consumer Messaging Semantics

Welcome to the session 6 lab 4. The work for this lab is done in `~/kafka-training/lab6.4`. In this lab, you are going to implement ***Exactly Once*** messaging semantics.

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

Find the latest version of this lab here.

Lab Exactly Once Semantics

To implement Exactly-Once semantics, you have to control and store the offsets for the partitions with the output of your consumer operation. You then have to read the stored positions when your consumer is assigned partitions to consume.

Remember that consumers do not have to use Kafka's built-in offset storage, and to implement ***exactly once messaging semantics***, you will need to read the offsets from stable storage. In this example, we use a JDBC database.

You will need to store offsets with processed record output to make it "exactly once" message consumption.

You will store the output of record consumption in an RDBMS with the offset, and partition. This approach allows committing both processed record output and location (partition/offset of record) in a single transaction thus implementing "exactly once" messaging.

StockPriceRecord

StockPriceRecord holds offset and partition info

~/kafka-training/lab6.4/src/main/java/com/cloudurable/kafka/consumer/StockPriceRecord.java

Kafka Consumer: StockPriceRecord

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

import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.TopicPartition;

public class StockPriceRecord  {

    private final String topic;
    private final int partition;
    private final long offset;
    private final String name;
    private final int dollars;
    private final int cents;
    private final boolean saved;
    private final TopicPartition topicPartition;

    public StockPriceRecord(String topic, int partition, long offset,
                            String name, int dollars, int cents, boolean saved) {
        this.topic = topic;
        this.partition = partition;
        this.offset = offset;
        this.name = name;
        this.dollars = dollars;
        this.cents = cents;
        this.saved = saved;
        topicPartition = new TopicPartition(topic, partition);
    }
...
}
```

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

DatabaseUtilities

The `DatabaseUtilities` class saves Topic, Offset, Partition data in the Database.

~/kafka-training/lab6.4/src/main/java/com/cloudurable/kafka/consumer/DatabaseUtilities.java

Kafka Consumer: DatabaseUtilities.saveStockPrice

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

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DatabaseUtilities {
...
    public static void saveStockPrice(final StockPriceRecord stockRecord,
```

```
                                    final Connection connection) throws SQLException {

        final PreparedStatement preparedStatement = getUpsertPreparedStatement(
                stockRecord.getName(), connection);



        //Save partition, offset and topic in database.
        preparedStatement.setLong( parameterindex:1, stockRecord.getOffset());
        preparedStatement.setLong( parameterindex:2, stockRecord.getPartition());
        preparedStatement.setString( parameterindex:3, stockRecord.getTopic());

        //Save stock price, name, dollars, and cents into database.
        preparedStatement.setInt( parameterindex:4, stockRecord.getDollars());
        preparedStatement.setInt( parameterindex:5, stockRecord.getCents());
        preparedStatement.setString( parameterindex:6, stockRecord.getName());

        //Save the record with offset, partition, and topic.
        preparedStatement.execute();

    }
...
}
```

To get exactly once, you need to save the offset and partition with the output of the consumer process.

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

SimpleStockPriceConsumer

"Exactly-Once" - Delivery Semantics

~/kafka-training/lab6.4/src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java

Kafka Consumer: SimpleStockPriceConsumer.pollRecordsAndProcess

```
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 org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

import static com.cloudurable.kafka.consumer.DatabaseUtilities.getConnection;
import static com.cloudurable.kafka.consumer.DatabaseUtilities.saveStockPrice;
import static com.cloudurable.kafka.consumer.DatabaseUtilities.startJdbcTransaction;

public class SimpleStockPriceConsumer
{
...
    private static void pollRecordsAndProcess(
            int readCountStatusUpdate,
            Consumer<String, StockPrice> consumer,
            Map<String, StockPriceRecord> currentStocks, int readCount) throws Exception {

        final ConsumerRecords<String, StockPrice> consumerRecords =
                consumer.poll(1000);

        if (consumerRecords.count() == 0) return;


        //Get rid of duplicates and keep only the latest record.
        consumerRecords.forEach(record -> currentStocks.put(record.key(),
                new StockPriceRecord(record.value(), saved: false, record)));

        final Connection connection = getConnection();
        try {
            startJdbcTransaction(connection)            //Start DB Transaction
            for (StockPriceRecord stockRecordPair : currentStocks.values()) {
```

```
                if (!stockRecordPair.isSaved()) {
                            //Save the record
                            //with partition/offset to DB.
                    saveStockPrice(stockRecordPair, connection);
                    //Mark the record as saved
                    currentStocks.put(stockRecordPair.getName(), new
                            StockPriceRecord(stockRecordPair, saved: true));
                }
            }
            connection.commit();            //Commit DB Transaction
            consumer.commitSync();            //Commit the Kafka offset
        } catch (CommitFailedException ex) {
            logger.error("Failed to commit sync to log", ex);
            connection.rollback();                    //Rollback Transaction
        } catch (SQLException sqle) {
            logger.error("Failed to write to DB", sqle);
            connection.rollback();                    //Rollback Transaction
        } finally {
            connection.close();
        }

        if (readCount % readCountStatusUpdate == 0) {
            displayRecordsStatsAndStocks(currentStocks, consumerRecords);
        }
    }
...
}
```

Try to commit the DB transaction and if it succeeds, commit the offset position.

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java` follow the instructions in the file to commit database transaction and Kafka log.

Initializing and saving offsets from ConsumerRebalanceListener

If implementing "exactly once" message semantics, then you have to manage offset positioning with a `ConsumerRebalanceListener` which gets notified when partitions are assigned or taken away from a consumer.

You will implement a `ConsumerRebalanceListener` and then pass the `ConsumerRebalanceListener` instance in call to `kafkaConsumer.subscribe(Collection, ConsumerRebalanceListener)`.

`ConsumerRebalanceListener` is notified when partitions get taken away from a consumer, so the consumer can commit its offset for partitions by implementing `ConsumerRebalanceListener.onPartitionsRevoked(Collection)`.

When partitions get assigned to a consumer, you will need to look up the offset in a database for new partitions and correctly initialize consumer to that position by implementing `ConsumerRebalanceListener.onPartitionsAssigned(Collection)`.

SeekToLatestRecordsConsumerRebalanceListener

"Exactly-Once" - Delivery Semantics

~/kafka-training/lab6.4/src/main/java/com/cloudurable/kafka/consumer/SeekToLatestRecordsConsumerRebalanceListener.java

Kafka Consumer: SeekToLatestRecordsConsumerRebalanceListener.onPartitionsAssigned

```
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 org.slf4j.Logger;

import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static org.slf4j.LoggerFactory.getLogger;

public class SeekToLatestRecordsConsumerRebalanceListener
                                implements ConsumeRebalanceListener{
    private final Consumer<String, StockPrice> consumer;
    private static final Logger logger = getLogger(SimpleStockPriceConsumer.class);

    public SeekToLatestRecordsConsumerRebalanceListener(
            final Consumer<String, StockPrice> consumer) {
        this.consumer = consumer;
    }
```

```
    @Override
    public void onPartitionsAssigned(final Collection<TopicPartition> partitions) {
        final Map<TopicPartition, Long> maxOffsets = getMaxOffsetsFromDatabase();
        maxOffsets.entrySet().forEach(
                entry -> partitions.forEach(topicPartition -> {
                    if (entry.getKey().equals(topicPartition)) {
                        long maxOffset = entry.getValue();
                        //Call to consumer.seek to move to the partition.
                        consumer.seek(topicPartition, offset: maxOffset + 1);
                        displaySeekInfo(topicPartition, maxOffset);
                    }
                }));
    }

    ...

    private Map<TopicPartition, Long> getMaxOffsetsFromDatabase() {
        final List<StockPriceRecord> records = DatabaseUtilities.readDB();
        final Map<TopicPartition, Long> maxOffsets = new HashMap<>();
        records.forEach(stockPriceRecord -> {
            final Long offset = maxOffsets.getOrDefault(stockPriceRecord.getTopicPartition(),
                    defaultValue: -1L);
            if (stockPriceRecord.getOffset() > offset) {
                maxOffsets.put(stockPriceRecord.getTopicPartition(),
                        stockPriceRecord.getOffset());
            }
        });
        return maxOffsets;
    }
...
```

We load a map of max offset per TopicPartition from the database. We could (should) use SQL, but for this example, we just use a map and iterate through the current stock price records looking for max. The `maxOffsets` key is TopicPartition and value is the max offset for that partition. Then we seek to that position with consumer.seek

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/SeekToLatestRecordsConsumerRebalanceListene`

follow the instructions in the file.

SimpleStockPriceConsumer

Subscribe to this topic using SeekToLatestRecordsConsumerRebalanceListener

~/kafka-training/lab6.4/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 org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

import static com.cloudurable.kafka.consumer.DatabaseUtilities.getConnection;
import static com.cloudurable.kafka.consumer.DatabaseUtilities.saveStockPrice;
import static com.cloudurable.kafka.consumer.DatabaseUtilities.startJdbcTransaction;

public class SimpleStockPriceConsumer
{
...

    private static void runConsumer(final int readCountStatusUpdate) throws InterruptedException {
        final Map<String, StockPriceRecord> map = new HashMap<>();


        try (final Consumer<String, StockPrice> consumer = createConsumer()) {

            consumer.subscribe(Collections.singletonList(StockAppConstants.TOPIC),
```

```
                    new SeekToLatestRecordsConsumerRebalanceListener(consumer));

            int readCount = 0;
            while (true) {
                try {
                    pollRecordsAndProcess(readCountStatusUpdate, consumer, map, readCount);
                } catch (Exception e) {
                    logger.error("Problem handling record processing", e);
                }
                readCount ++;
            }
        }
    }

...
}
```

ACTION - EDIT `src/main/java/com/cloudurable/kafka/consumer/SimpleStockPriceConsumer.java` method and follow the instructions to subscribe to topic using SeekToLatestRecordsConsumerRebalanceListener.

ACTION - RUN ZooKeeper and Brokers if needed.

ACTION - RUN SimpleStockPriceConsumer from IDE

ACTION - RUN StockPriceKafkaProducer from IDE

ACTION - OBSERVE and then STOP consumer and producer

Expected behavior

You should see offset messages from `SeekToLatestRecordsConsumerRebalanceListener` in the log for the consumer.

ACTION - STOP SimpleStockPriceConsumer from IDE (while you leave StockPriceKafkaProducer for 30 seconds)

ACTION - RUN SimpleStockPriceConsumer from IDE

Expected behavior

Again, you should see offset messages from `SeekToLatestRecordsConsumerRebalanceListener` in the log for the consumer.

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