# Lab 6.5: Thread per Consumer

Welcome to the session 6 lab 5. The work for this lab is done in `~/kafka-training/labs/lab6.5` . In this lab, you are going to implement a thread per consumer.

Find the latest version of this lab .

## Lab Thread per consumer

Unlike Kafka producers, Kafka consumers are not thread-safe.

All network I/O happens in a thread of the application making calls. Kafka Consumers manage buffers, and connections state that threads can't share.

The only exception thread-safe method that the consumer has is `consumer.wakeup()` . The `wakeup()` method forces the consumer to throw a `WakeupException` on any thread the consumer client is blocking. You can use this to shut down a consumer from another thread.

## Consumer per thread

The easiest to implement a client application that can handle more work is to use a thread per consumer and then spin up more consumers. This approach works best because it requires no inter-thread co-ordination. You don't have to worry about in-order processing on a per-partition basis because Kafka is already sending messages by key to the partitions that you are managing so in-order processing is natural. This approach is easy to implement. Just process records in the order that you receive them.

## StockPriceConsumerRunnable is Runnable

To create a consumer per thread, we will move away from our `SimpleStockPriceConsumer` and use a new class called `StockPriceConsumerRunnable` that implements Runnable. We will then use a thread pool to launch `StockPriceConsumerRunnable` instances.

**~/kafka-training/labs/lab6.5/src/main/java/com/cloudurable/kafka/consumer/StockPriceConsumerRunnable.java**

**Kafka Consumer: StockPriceConsumerRunnable**

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


import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.clients.consumer.CommitFailedException;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.atomic.AtomicBoolean;
```

```java
import static com.cloudurable.kafka.StockAppConstants.TOPIC;

public class StockPriceConsumerRunnable implements Runnable{
    private static final Logger logger =
            LoggerFactory.getLogger(StockPriceConsumerRunnable.class);

    private final Consumer<String, StockPrice> consumer;
    private final int readCountStatusUpdate;
    private final int threadIndex;
    private final AtomicBoolean stopAll;
    private boolean running = true;


    @Override
    public void run() {
        try {
        runConsumer();
        } catch (Exception ex) {
        logger.error("Run Consumer Exited with", ex);
        }
    }


    ...


    void runConsumer() throws Exception {
        // Subscribe to the topic.
        consumer.subscribe(Collections.singletonList(TOPIC));
        final Map<String, StockPrice> lastRecordPerStock = new HashMap<>();
        try {
            int readCount = 0;
            while (isRunning()) {
                pollRecordsAndProcess(lastRecordPerStock, readCount);
            }
        } finally {
            consumer.close();
        }
    }


    private void pollRecordsAndProcess(
            final Map<String, StockPrice> currentStocks,
            final int readCount) throws Exception {

        final ConsumerRecords<String, StockPrice> consumerRecords =
                consumer.poll(100);

        if (consumerRecords.count() == 0) {
            if (stopAll.get()) this.setRunning(false);
            return;
        }
        consumerRecords.forEach(record -> currentStocks.put(record.key()
                new StockPriceRecord(record.value(), saved: true, record)));
```

```
        try {
            startTransaction();                        //Start DB Transaction

            processRecords(currentStocks, consumerRecords);
            consumer.commitSync();                     //Commit the Kafka offset
            commitTransaction();                       //Commit DB Transaction
        } catch (CommitFailedException ex) {
            logger.error("Failed to commit sync to log", ex);
            rollbackTransaction();                     //Rollback Transaction
        }


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

## ConsumerMain

We will also create a `ConsumerMain` class that will start up thread pool. It will create a producer per thread. Then it will submit the producers (StockPriceConsumerRunnable, which are runnable) to the `executorService` (`threadPool`).

**~/kafka-training/labs/lab6.5/src/main/java/com/cloudurable/kafka/consumer/ConsumerMain.java**

**Kafka Consumer: ConsumerMain**

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

import com.cloudurable.kafka.StockAppConstants;
import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Properties;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.stream.IntStream;

import static java.util.concurrent.Executors.newFixedThreadPool;

public class ConsumerMain {
    ...
    public static void main(String... args) throws Exception {
        final int threadCount = 5;
```

```
        final ExecutorService executorService = newFixedThreadPool(threadCount);
        final AtomicBoolean stopAll = new AtomicBoolean();

        IntStream.range(0, threadCount).forEach(index -> {
            final StockPriceConsumerRunnable stockPriceConsumer =
                new StockPriceConsumerRunnable(createConsumer(),
                        readCountStatusUpdate: 10, index, stopAll);
            executorService.submit(stockPriceConsumer);
        });
        ...
    }
    ...
}
```

**Lab Work**

*ACTION* - **EDIT**
`com.cloudurable.kafka.consumer.StockPriceConsumerRunnable`
**and follow the instructions in the file.**

*ACTION* - **EDIT** `com.cloudurable.kafka.consumer.ConsumerMain`
**and follow the instructions in the file.**

*ACTION* - **RUN ZooKeeper and Brokers if needed.**

*ACTION* - **RUN ConsumerMain from IDE**

*ACTION* - **RUN StockPriceKafkaProducer from IDE**

*ACTION* - **OBSERVE and then STOP consumers and producer**

**Expected behavior**

It should run and should get messages like this:

**Expected output**

```
New ConsumerRecords par count 1 count 3, max offset
ticker AAA price 80.25 Thread 1
ticker CCC price 80.25 Thread 1
ticker EEE price 80.25 Thread 1
ticker DEF price 94.44 Thread 1
ticker XYZ price 94.44 Thread 1

New ConsumerRecords par count 1 count 2, max offset
ticker IBM price 61.74 Thread 2
ticker UBER price 544.94 Thread 2

New ConsumerRecords par count 1 count 3, max offset
```

```
ticker GOOG price 448.74 Thread 0
ticker ABC price 94.44 Thread 0
ticker BBB price 80.25 Thread 0
ticker DDD price 80.25 Thread 0
ticker FFF price 80.25 Thread 0
ticker SUN price 61.74 Thread 0
ticker INEL price 61.74 Thread 0
```

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