Lab 6.7: Implementing a priority queue with consumer.assign()

Welcome to the session 6 lab 7. The work for this lab is done in `~/kafka-training/lab6.7`. In this lab, you are going to implement a priority queue with `consumer.assign()`.

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

Find the latest version of this lab here.

Lab Using consumer.assign to implement a priority queue

In this lab, you will implement a priority processing queue. You will use `consumer.partitionsFor(TOPIC)` to get a list of partitions. Usage like this simplest when the partition assignment is also done manually using `assign()` instead of `subscribe()`. Use `assign()`, pass a `TopicPartition` from the consumer worker. Use the `Partitioner` from an earlier example for Producer so only important stocks get sent to the important partition.

Using partitionsFor() for Priority Queue

~/kafka-training/lab6.7/src/main/java/com/cloudurable/kafka/consumer/ConsumerMain.java

Kafka Consumer: ConsumerMain.main

```java
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.ArrayList;
import java.util.List;
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 com.cloudurable.kafka.StockAppConstants.TOPIC;
import static java.util.concurrent.Executors.newFixedThreadPool;

public class ConsumerMain {
    ...
    public static void main(String... args) throws Exception {
        final AtomicBoolean stopAll = new AtomicBoolean();
        final Consumer<String, StockPrice> consumer = createConsumer();

        //Get the partitions
        final List<PartitionInfo> partitionInfos = consumer.partitionsFor(TOPIC);

        final int threadCount = partitionInfos.size();
        final int numWorkers = 5;
        final ExecutorService executorService = newFixedThreadPool(threadCount);

        IntStream.range(0, threadCount).forEach(index -> {
            final PartitionInfo partitionInfo = partitionInfos.get(index);
            final boolean leader = partitionInfo.partition() == partitionInfos.size() -1;
            final int workerCount = leader ? numWorkers * 3 : numWorkers;
            final StockPriceConsumerRunnable stockPriceConsumer =
                    new StockPriceConsumerRunnable(partitionInfo, createConsumer(),
                            readCountStatusUpdate: 10, index, stopAll, workerCount);
            consumerList.add(consumer);
            executorService.submit(stockPriceConsumer);
        });
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            logger.info("Stopping app");
            stopAll.set(true);
            sleep();
            consumerList.forEach(Consumer::wakeup);
            executorService.shutdown();
            try {
                executorService.awaitTermination(5_000, TimeUnit.MILLISECONDS);
                if (!executorService.isShutdown())
                    executorService.shutdownNow();
            } catch (InterruptedException e) {
                logger.warn("shutting down", e);
            }
            sleep();
            consumerList.forEach(Consumer::close);
        }));
    }
...
}
```

Notice that the index is the topic partition. Num threads are the partition count, and the priority partition gets extra workers.

Using assign() for Priority Queue

~/kafka-training/lab6.7/src/main/java/com/cloudurable/kafka/consumer/StockPriceConsumerRunnable.java

Kafka Consumer: StockPriceConsumerRunnable.runConsumer

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

import com.cloudurable.kafka.model.StockPrice;
import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.TopicPartition;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collections;
import java.util.Map;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicBoolean;

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


public class StockPriceConsumerRunnable implements Runnable {
    ...
    private void runConsumer() throws Exception {
        //Assign a partition
        consumer.assign(Collections.singleton(topicPartition));
        final Map<String, StockPrice> lastRecordPerStock = new HashMap<>();
        try {
                int readCount = 0;
                while (isRunning()) {
                    pollRecordsAndProcess(lastRecordPerStock, readCount);
                }
        } finally {
            consumer.close();
        }
    }
    ...
}
```

Lab Work

Use the slides for Session 6 as a guide.

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

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

ACTION **- RECREATE the topic with five partitions (HINT: bin/create-topic.sh) and use 5 partitions.**

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 153, max offset
ticker IBM price 66.59 Thread 4
ticker UBER price 241.94 Thread 4

New ConsumerRecords par count 1 count 220, max offset
ticker ABC price 95.85 Thread 2
ticker BBB price 53.36 Thread 2
ticker FFF price 70.34 Thread 2

New ConsumerRecords par count 1 count 318, max offset
ticker GOOG price 458.44 Thread 0
ticker DDD price 68.38 Thread 0
ticker SUN price 91.90 Thread 0
ticker INEL price 65.94 Thread 0

New ConsumerRecords par count 1 count 364, max offset
ticker AAA price 66.53 Thread 1
ticker DEF price 65.94 Thread 1
ticker EEE price 70.34 Thread 1
ticker XYZ price 65.94 Thread 1
```

Try the following

Try using different worker pool sizes and different consumer thread pool sizes. Try adding a small wait for the processing. Try 10ms. It should all run. Stop consumer and producer when finished.

Kafka Tutorial

This comprehensive *Kafka tutorial* covers Kafka architecture and design. The *Kafka tutorial* has example Java Kafka producers and Kafka consumers. The *Kafka tutorial* also covers Avro and Schema Registry.

Complete Kafka Tutorial: Architecture, Design, DevOps and Java Examples.

- Kafka Tutorial Part 1: What is Kafka?
- Kafka Tutorial Part 2: Kafka Architecture
- Kafka Tutorial Part 3: Kafka Topic Architecture
- Kafka Tutorial Part 4: Kafka Consumer Architecture
- Kafka Tutorial Part 5: Kafka Producer Architecture
- Kafka Tutorial Part 6: Using Kafka from the command line

- Kafka Tutorial Part 7: Kafka Broker Failover and Consumer Failover
- Kafka Tutorial Part 8: Kafka Ecosystem
- Kafka Tutorial Part 9: Kafka Low-Level Design
- Kafka Tutorial Part 10: Kafka Log Compaction Architecture
- Kafka Tutorial Part 11: Writing a Kafka Producer example in Java
- Kafka Tutorial Part 12: Writing a Kafka Consumer example in Java
- Kafka Tutorial Part 13: Writing Advanced Kafka Producer Java examples
- Kafka Tutorial 14: Writing Advanced Kafka Consumer Java examples
- Kafka Tutorial Part 15: Kafka and Avro
- Kafka Tutorial Part 16: Kafka and Schema Registry
- Kafka Tutorial

About Cloudurable

We hope you enjoyed this article. Please provide feedback. Cloudurable provides Kafka training, Kafka consulting, Kafka support and helps setting up Kafka clusters in AWS.