

## Lab 5.7: Kafka Interceptor



Welcome to the session 5 lab 7. The work for this lab is done in `~/kafka-training/labs/lab5.7`. In this lab, you are going to set up Kafka Producer interceptor.

### Producer Interception

You will configure our Producer config and set the config property: `interceptor.classes` to our `ProducerInterceptor` which we will define shortly. The `ProducerInterceptor` will print out debug information when we send a message and when the broker acknowledges a message. The interceptors we pass must implement `ProducerInterceptor` interface so we will define a `StockProducerInterceptor` that implements `ProducerInterceptor`. The `StockProducerInterceptor` will intercept records that the producer sends to broker and after intercept acks from the broker.

Let's define the `StockProducerInterceptor` as follows:

### KafkaProducer ProducerInterceptor

`~/kafka-training/labs/lab5.7/src/main/java/com/fenago/kafka/producer/StockProducerInterceptor.java`

**Kafka Producer: StockProducerInterceptor**

```
package com.fenago.kafka.producer;

import org.apache.kafka.clients.producer.ProducerInterceptor;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Map;

public class StockProducerInterceptor implements ProducerInterceptor {
    ...
}
```

Notice that the `StockProducerInterceptor` implements `ProducerInterceptor`.

**ACTION** - EDIT `StockProducerInterceptor.java` and change it to implement `ProducerInterceptor`.

### ProducerInterceptor onSend

The `onSend` method gets called before the record is sent to the broker.

`~/kafka-training/labs/lab5.7/src/main/java/com/fenago/kafka/producer/StockProducerInterceptor.java`

**Kafka Producer: StockProducerInterceptor onSend**

```
package com.fenago.kafka.producer;
...

public class StockProducerInterceptor implements ProducerInterceptor {
```

```

private final Logger logger = LoggerFactory
    .getLogger (StockProducerInterceptor.class);
private int onSendCount;
private int onAckCount;

@Override
public ProducerRecord onSend(final ProducerRecord record) {
    onSendCount++;
    if (logger.isDebugEnabled()) {
        logger.debug(String.format("onSend topic=%s key=%s value=%s %d \n",
            record.topic(), record.key(), record.value().toString(),
            record.partition()
        ));
    } else {
        if (onSendCount % 100 == 0) {
            logger.info(String.format("onSend topic=%s key=%s value=%s %d \n",
                record.topic(), record.key(), record.value().toString(),
                record.partition()
            ));
        }
    }
    return record;
}
}

```

The `StockProducerInterceptor` overrides the `onSend` method and increments `onSendCount`. Every 100 `onSendCount`, we print out record data.

**ACTION** - EDIT `StockProducerInterceptor.java` and implement the `onSend` method something like above.

## ProducerInterceptor onAck

The `onAck` method gets called after the broker acknowledges the record.

~/kafka-training/labs/lab5.7/src/main/java/com/fenago/kafka/producer/StockProducerInterceptor.java

**Kafka Producer: StockProducerInterceptor onAck**

```

package com.fenago.kafka.producer;
...

public class StockProducerInterceptor implements ProducerInterceptor {

    private final Logger logger = LoggerFactory
        .getLogger (StockProducerInterceptor.class);
    ...
    private int onAckCount;

    @Override
    public void onAcknowledgement(final RecordMetadata metadata,

```

```

        final Exception exception) {

    onAckCount++;

    if (logger.isDebugEnabled()) {
        logger.debug(String.format("onAck topic=%s, part=%d, offset=%d\n",
            metadata.topic(), metadata.partition(), metadata.offset()
        ));
    } else {
        if (onAckCount % 100 == 0) {
            logger.info(String.format("onAck topic=%s, part=%d, offset=%d\n",
                metadata.topic(), metadata.partition(), metadata.offset()
            ));
        }
    }
}
}
}

```

The StockProducerInterceptor overrides the onAck method and increments onAckCount. Every 100 onAckCount, we print out record data.

**ACTION** - EDIT StockProducerInterceptor.java and implement the `onAcknowledgement` method something like above.

## ProducerInterceptor the rest

There are other methods to override.

~/kafka-training/labs/lab5.7/src/main/java/com/fenago/kafka/producer/StockProducerInterceptor.java

## Kafka Producer: StockProducerInterceptor the rest

```

package com.fenago.kafka.producer;

import org.apache.kafka.clients.producer.ProducerInterceptor;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Map;

public class StockProducerInterceptor implements ProducerInterceptor {

    private final Logger logger = LoggerFactory
        .getLogger(StockProducerInterceptor.class);

    private int onSendCount;
    private int onAckCount;

    @Override
    public ProducerRecord onSend(final ProducerRecord record) {
        onSendCount++;
        if (logger.isDebugEnabled()) {

```

```

        logger.debug(String.format("onSend topic=%s key=%s value=%s %d \n",
            record.topic(), record.key(), record.value().toString(),
            record.partition()
        ));
    } else {
        if (onSendCount % 100 == 0) {
            logger.info(String.format("onSend topic=%s key=%s value=%s %d \n",
                record.topic(), record.key(), record.value().toString(),
                record.partition()
            ));
        }
    }
    return record;
}

@Override
public void onAcknowledgement(final RecordMetadata metadata,
    final Exception exception) {
    onAckCount++;

    if (logger.isDebugEnabled()) {
        logger.debug(String.format("onAck topic=%s, part=%d, offset=%d\n",
            metadata.topic(), metadata.partition(), metadata.offset()
        ));
    } else {
        if (onAckCount % 100 == 0) {
            logger.info(String.format("onAck topic=%s, part=%d, offset=%d\n",
                metadata.topic(), metadata.partition(), metadata.offset()
            ));
        }
    }
}

@Override
public void close() {
}

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

```

We have to override close and configure.

**ACTION** - EDIT StockProducerInterceptor.java and implement the `close` and `configure` methods.

Next we need to configure the StockProducerInterceptor in the StockPriceKafkaProducer producer config.

## KafkaProducer - Interceptor Config

~/kafka-training/labs/lab5.7/src/main/java/com/fenago/kafka/producer/StockPriceKafkaProducer.java

## Kafka Producer: StockPriceKafkaProducer

```
public class StockPriceKafkaProducer {

    private static Producer<String, StockPrice>
        createProducer() {

        final Properties props = new Properties();
        setupBootstrapAndSerializers(props);
        setupBatchingAndCompression(props);
        setupRetriesInFlightTimeout(props);

        //Install interceptor list - config "interceptor.classes"
        props.put(ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
            StockProducerInterceptor.class.getName());

        return new KafkaProducer<>(props);
    }
}
```

The above sets the `StockProducerInterceptor.class.getName()` in the config property

`ProducerConfig.INTERCEPTOR_CLASSES_CONFIG`.

**ACTION** - EDIT `StockPriceKafkaProducer.java` and configure `ProducerConfig.INTERCEPTOR_CLASSES_CONFIG` as described above.

## Run it. Run Servers. Run Producer. Results.

Next we startup ZooKeeper if needed, and start or restart Kafka brokers as before. Then run the `StockPriceKafkaProducer` and look for log message from `ProducerInterceptor` in output.

**ACTION** - START ZooKeeper and Kafka Brokers if needed

**ACTION** - RUN `StockPriceKafkaProducer` from the IDE

**ACTION** - RUN `SimpleStockPriceConsumer` from the IDE

**ACTION** - LOOK for `onAck` and `onSend` messages in the `StockPriceKafkaProducer` log.

## Results ProducerInterceptor Output

You should see `oAck` and `onSend` messages in the log from the interceptor.