Lab 04 Consuming Messages from Kafka

a. Kafka Consumer (Java, C#, Python)

The goal of this lab is to create a simple Kafka consumer, that consumes records from the `driver-positions` topic.

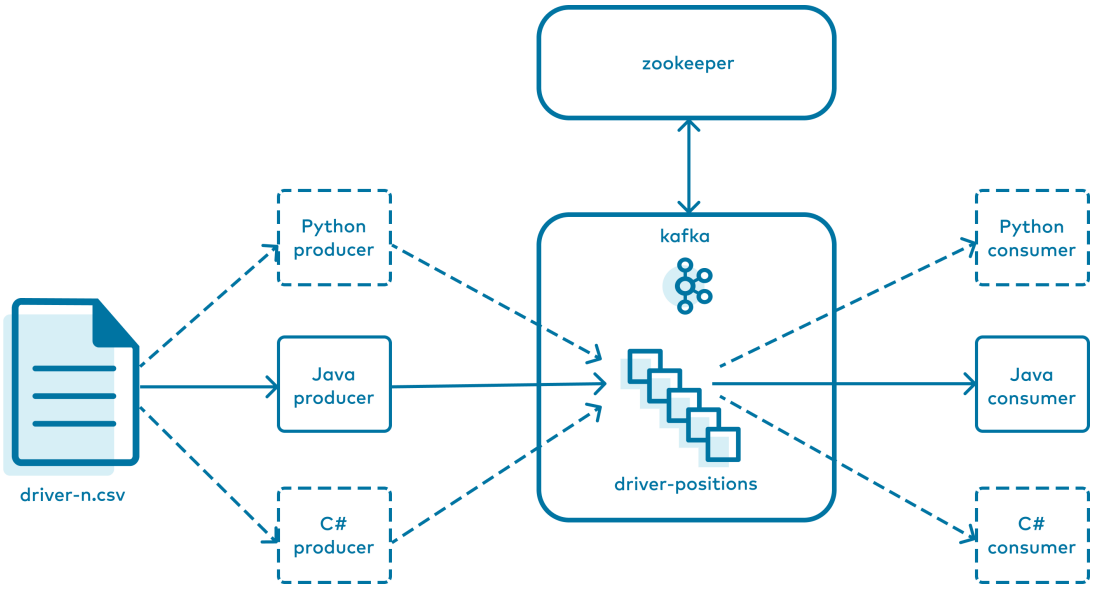
Prerequisites

1. Use the command in the table below to navigate to the project folder for your language. Click the associated hyperlink to open the API reference:

Language	Command	API Reference
Java	`cd ~/confluent-dev/challenge/java-consumer`	Class KafkaConsumer<K,V>
C#	`cd ~/confluent-dev/challenge/dotnet-consumer`	Interface IConsumer<TKey, TValue>
Python	`cd ~/confluent-dev/challenge/python-consumer`	class confluent_kafka.Consumer

2. Run the Kafka cluster:

```
$ docker-compose up -d zookeeper kafka control-center create-topics webserver
```



3. If you are completing the C# or Python exercise, install the dependencies.

   a. For C#:

   ```
   $ dotnet restore
   ```

   b. For Python:

   ```
   $ pip3 install -r requirements.txt
   ```

4. Open the project in Visual Studio Code:

   ```
   $ code .
   ```

Writing the Consumer

1. Run the producer solution from the previous exercise in a terminal window. This will give you live data in the `driver-positions` topic. From a terminal window run:

   ```
   $ cd ~/confluent-dev/solution/java-producer && \
     ./gradlew run --console plain
   ```

2. Open the implementation file for your language of choice

   - Java: `src/main/java/clients/Consumer.java`

- C#: `Program.cs`
- Python: `main.py`.

3. Locate the `TODO` comments in your implementation file. Use the API reference for your language to attempt each challenge. Solutions are provided at the end of this lab and in the `~/confluent-dev/solution` folder.

4. At any time run the application by selecting the menu **Run** → **Start Debugging** in VS Code. As you complete the challenges try to produce a similar output from your application:

```
Starting Java Consumer.
Key:driver-1 Value:47.618579,-122.355081 [partition 1]
Key:driver-1 Value:47.618577152452055,-122.35520620652974 [partition
1]
Key:driver-1 Value:47.61857902704408,-122.35507321130525 [partition
1]
Key:driver-1 Value:47.618579488930855,-122.35494018791431 [partition
1]
Key:driver-1 Value:47.61857995081763,-122.35480716452278 [partition
1]
...
```

5. Leave your consumer running. In a terminal window run this command to inspect the status of your consumer group:

```
$ kafka-consumer-groups \
    --bootstrap-server kafka:9092 \
    --describe \
    --group java-consumer \
    --group csharp-consumer \
    --group python-consumer

GROUP           TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-
OFFSET  LAG ...
csharp-consumer driver-positions 0          -               0
-
csharp-consumer driver-positions 1          2148            2153
5
csharp-consumer driver-positions 2          -               0
-
```

You can see some interesting metrics for your topic consumption. `CURRENT-OFFSET` is the last *committed* offset from your consumers. `LOG-END-OFFSET` is the last offset in each partition. `LAG` is *how far behind* the consumption is, or in other words `LOG-END-OFFSET -`

`CURRENT-OFFSET`. These metrics are very useful when checking if your consumption is keeping up with production.

6. When you have completed the challenges, stop the debugger in VS Code.
7. Return to the terminal window running the producer solution. Press `Ctrl+C` to exit the producer.

Extra Challenges and Questions

1. End your processing, and launch the consumer again. You'll see that the second time you run the application processing begins from a non-zero offset. Does `auto.offset.reset` apply the second time the application is run?
2. How do consumers know where to begin their processing?
3. Can you think of a way to make your next run of the application begin at the offset at the start of each partition?
4. Experiment with consumer settings of `fetch.max.wait.ms` (default: 500ms) and `fetch.min.bytes` (default: 1 byte). How would you expect the consumer to behave with the settings below?

   a. Java

   ```
   settings.put(ConsumerConfig.FETCH_MAX_WAIT_MS_CONFIG, "5000");
   settings.put(ConsumerConfig.FETCH_MIN_BYTES_CONFIG, "5000000");
   ```

   b. C#

   ```
   FetchWaitMaxMs = 5000,
   FetchMinBytes = 5000000,
   ```

   c. Python

   ```
   "fetch.wait.max.ms": "5000",
   "fetch.min.bytes": "5000000"
   ```

Java Solution

solution/java-consumer/src/main/java/clients/Consumer.java

```
// TODO: Poll for available records
final ConsumerRecords<String, String> records = consumer.poll(Duration
.ofMillis(100));
```

```
// TODO: print the contents of the record
System.out.printf("Key:%s Value:%s [partition %s]\n",
    record.key(), record.value(), record.partition());
```

C# Solution

solution/dotnet-consumer/Program.cs

```
// TODO: Consume available records
var cr = consumer.Consume(cts.Token);
```

```
// TODO: print the contents of the record
Console.WriteLine($"Key:{cr.Message.Key} Value:{cr.Message.Value}
[partition {cr.Partition.Value}]");
```

Python Solution

solution/python-consumer/main.py

```
#TODO: Poll for available records
msg = consumer.poll(1.0)
```

```
#TODO: print the contents of the record
print("Key:{} Value:{} [partition {}]".format(
    msg.key().decode('utf-8'),
    msg.value().decode('utf-8'),
    msg.partition()
))
```

Extra Challenges and Questions Solutions

1. `auto.offset.reset` would not be used on the second launch of your consumer. `auto.offset.reset` is used when there is no committed position (e.g. the group is first

initialized) or when an offset is out of range.

2. An instance in a consumer group sends its offset commits and fetches to a group coordinator broker. The group coordinators read from and write to special compacted Kafka topic named *__consumer_offsets*.

   Curious about the *__consumer_offsets* topic? You can consume message on this topic while your consumer runs with the command below:

```
$ kafka-console-consumer --bootstrap-server kafka:9092 \
--topic __consumer_offsets \
--formatter
"kafka.coordinator.group.GroupMetadataManager\$OffsetsMessageFormatte
r" \
| grep 'driver-positions'
```

3. You could begin consumption at the start of each partition simply by updating your consumer to use a new `GROUP_ID`. Also, the utility `kafka-consumer-groups` has a parameter `--to-earliest` which will set offsets to earliest offset. In the next exercise we will see how to programmatically *seek* to an offset for consumption.

4. With the supplied settings for `fetch.max.wait.ms` and `fetch.min.bytes` we see results roughly every 5 seconds. From the consumer documentation:

   > fetch.max.wait.ms: The maximum amount of time the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy the requirement given by fetch.min.bytes.

STOP HERE. THIS IS THE END OF THE EXERCISE.