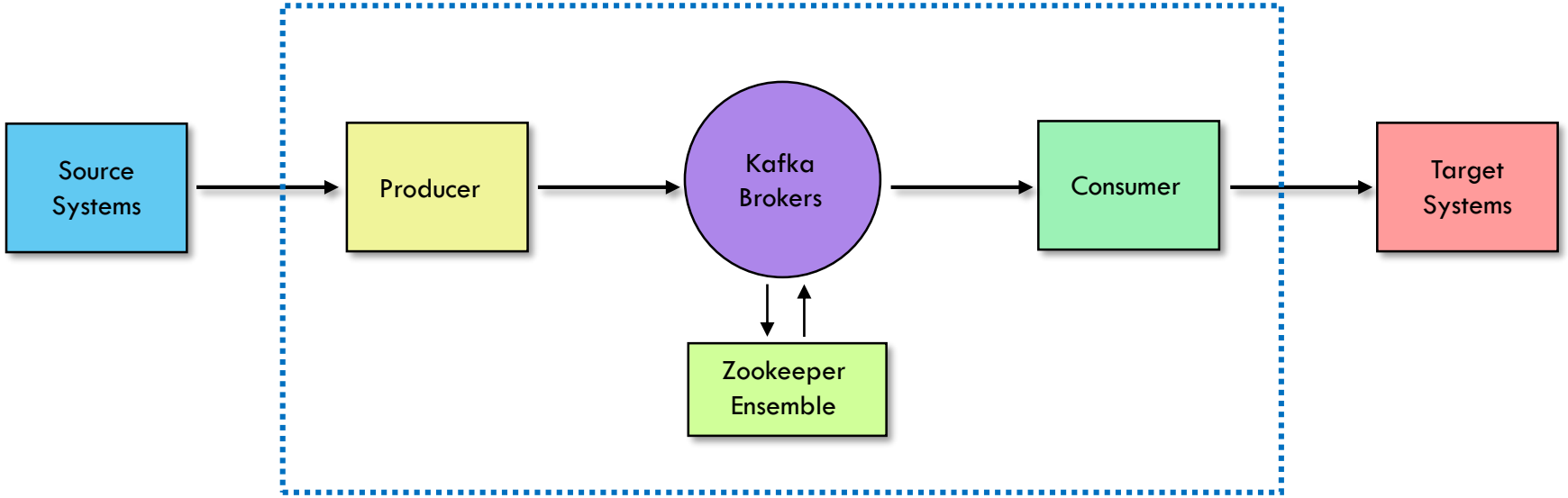
ak系列 – ak03

(Python Producer & Consumer – 進階)

Kafka Ecosystem:
Kafka Core



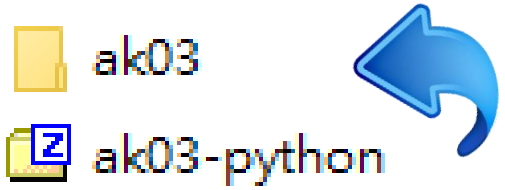
Download Sample Python Project Code

- Get Python Demo source code

Decompress Demo Python

在"hands-on"下新增一個"python"的檔案目錄

1



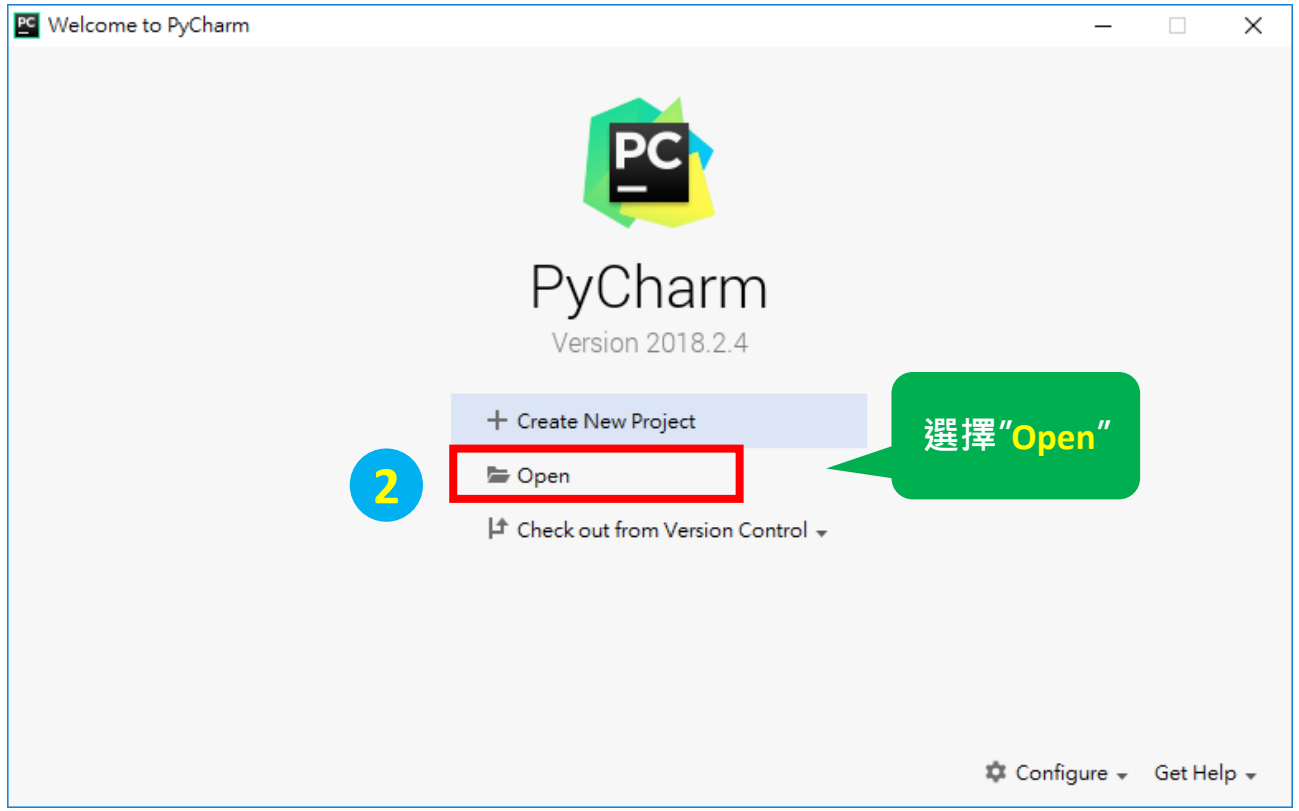
2

下載"ak03-python.zip"的package, 並解壓縮!

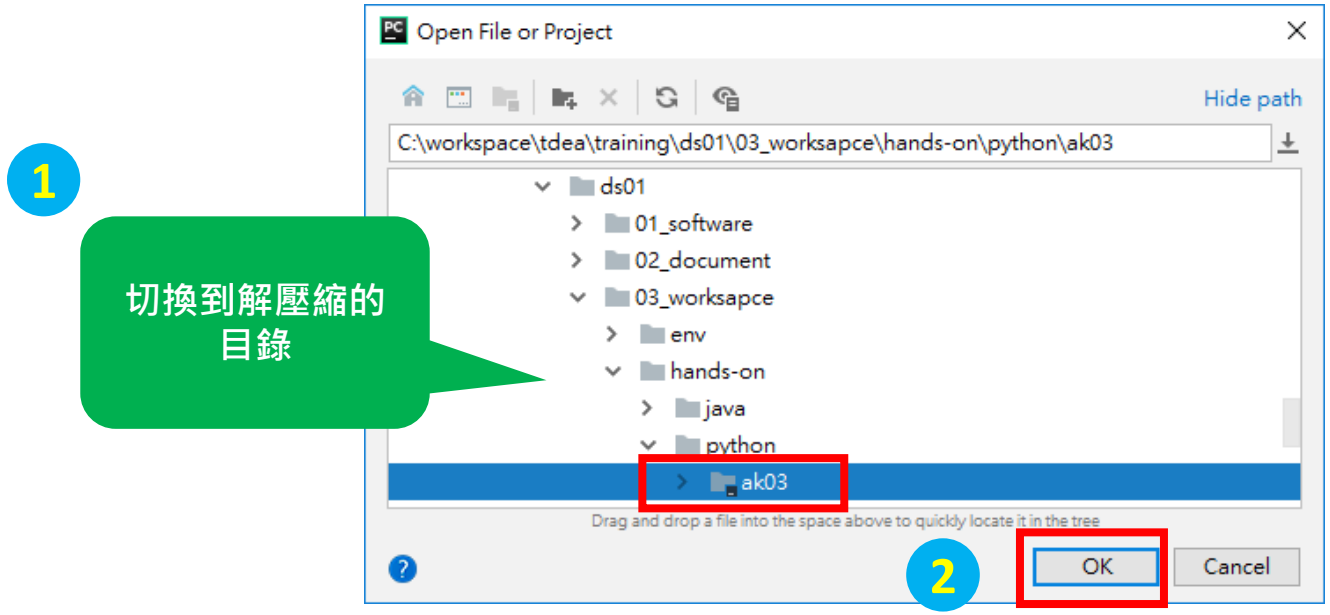
Open Demo Python Project using **PyCharm**

1

啟動 **PyCharm**

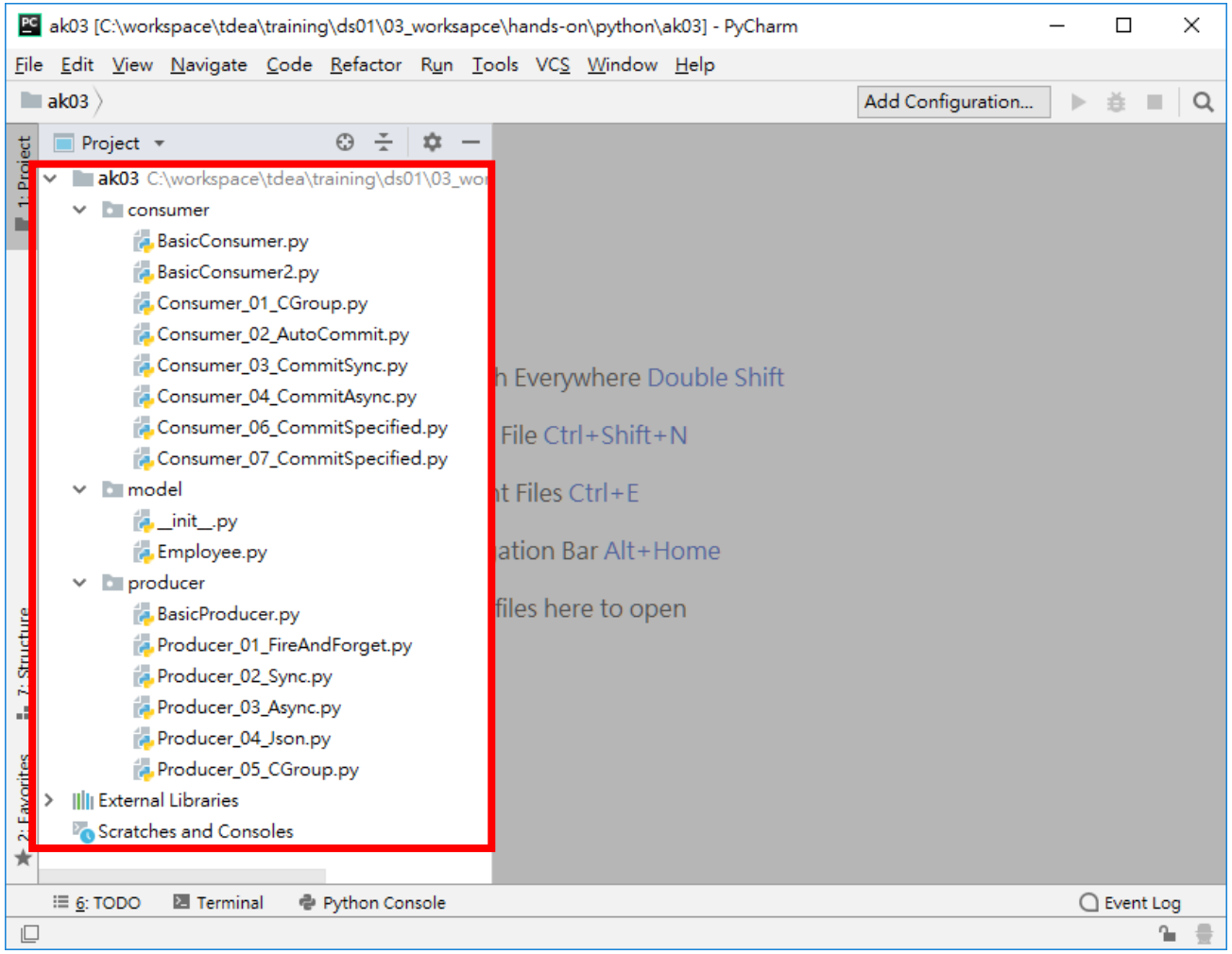


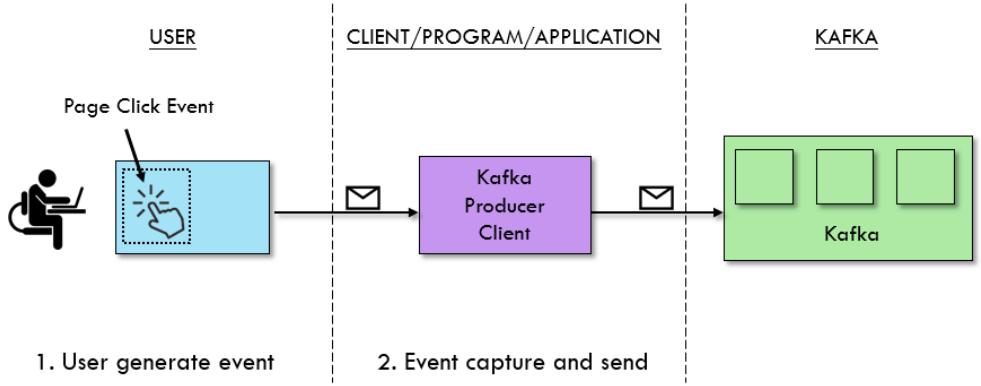
Open Demo Python Project using **PyCharm**



Open Demo Python Project using **PyCharm**

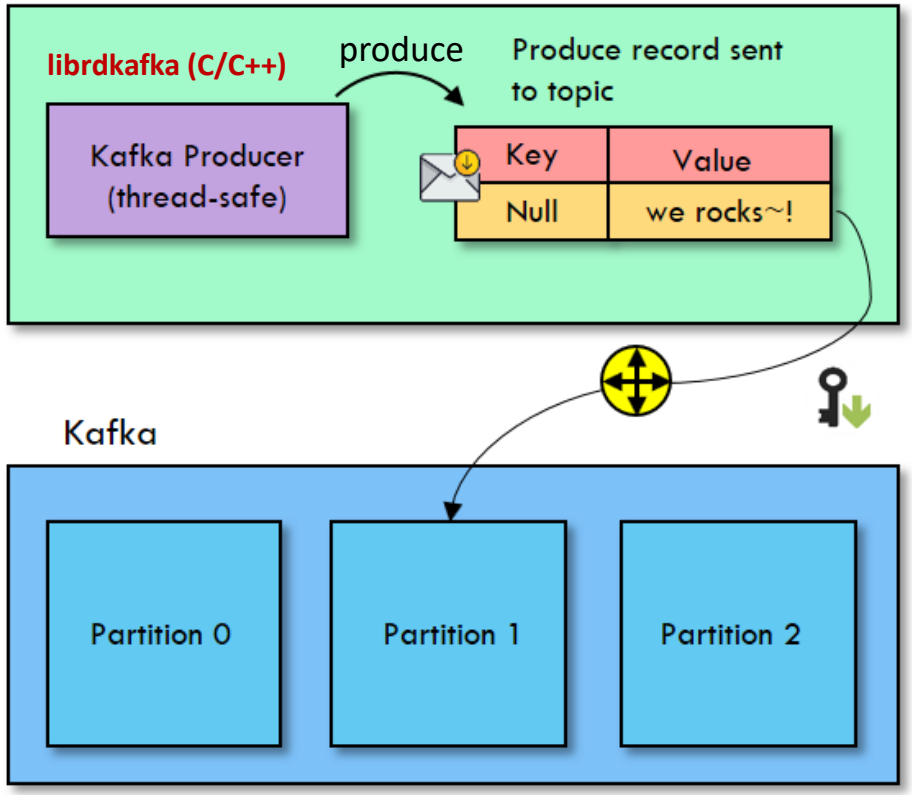
1

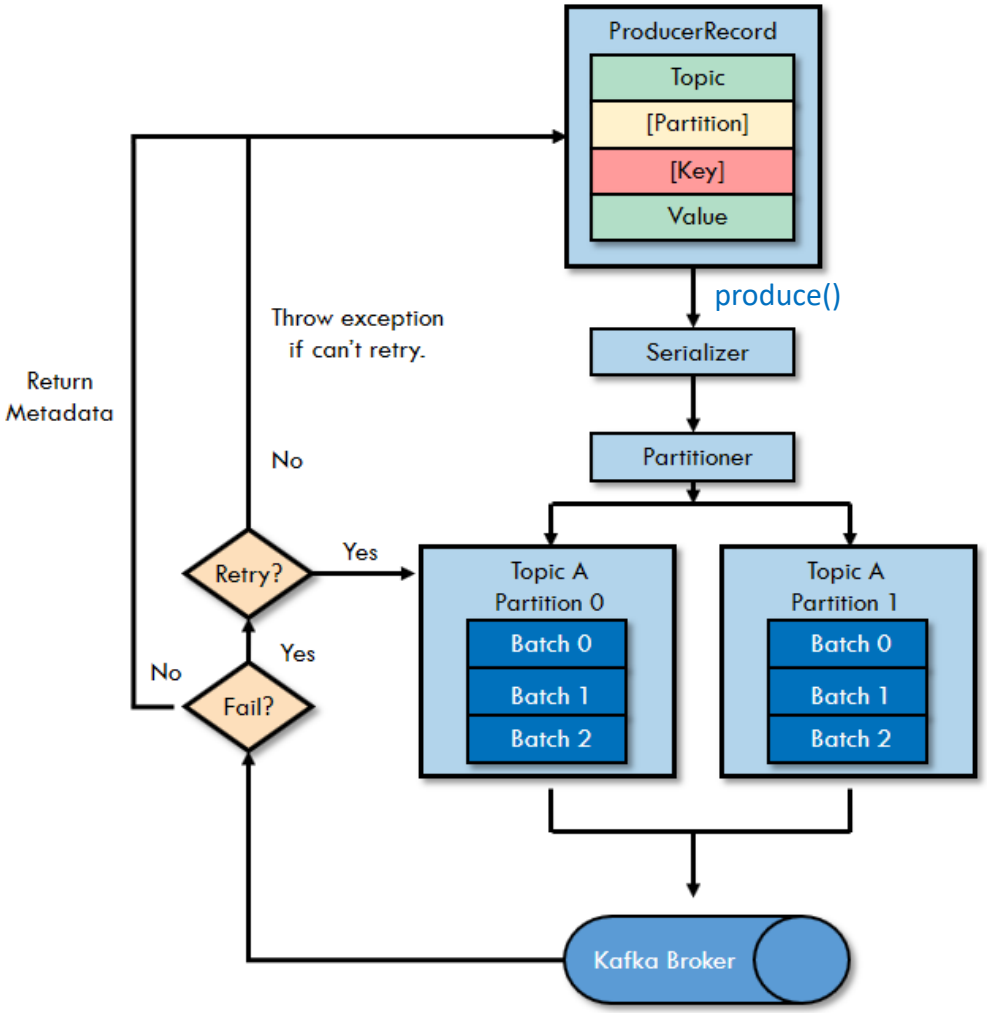




Kafka Producers: Writing Messages to Kafka

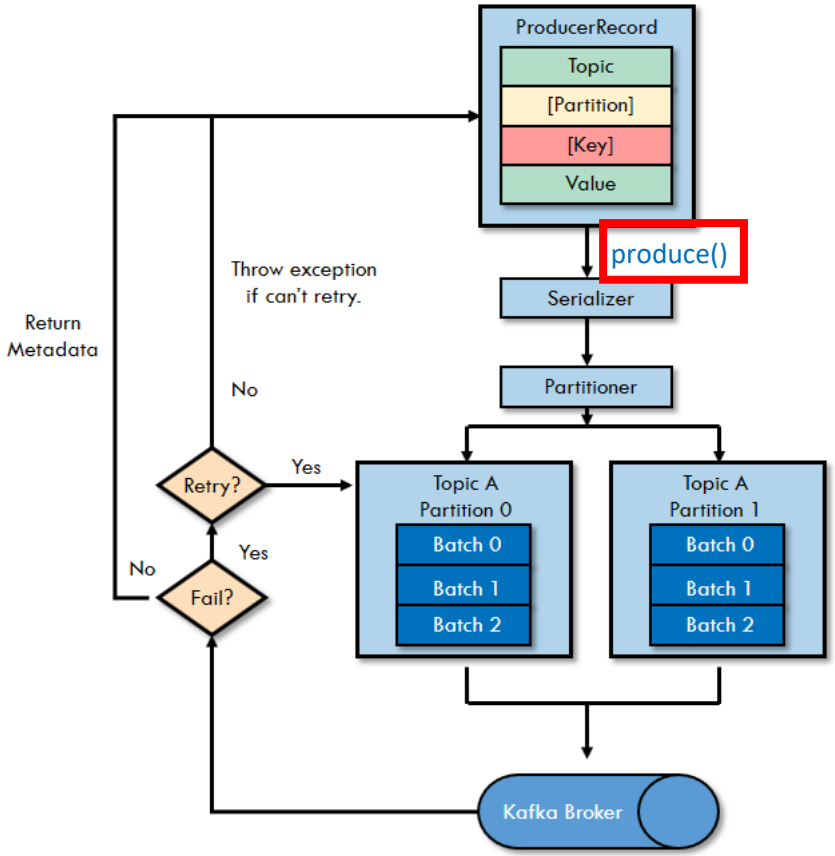
Producer Overview





Three primary methods of sending messages

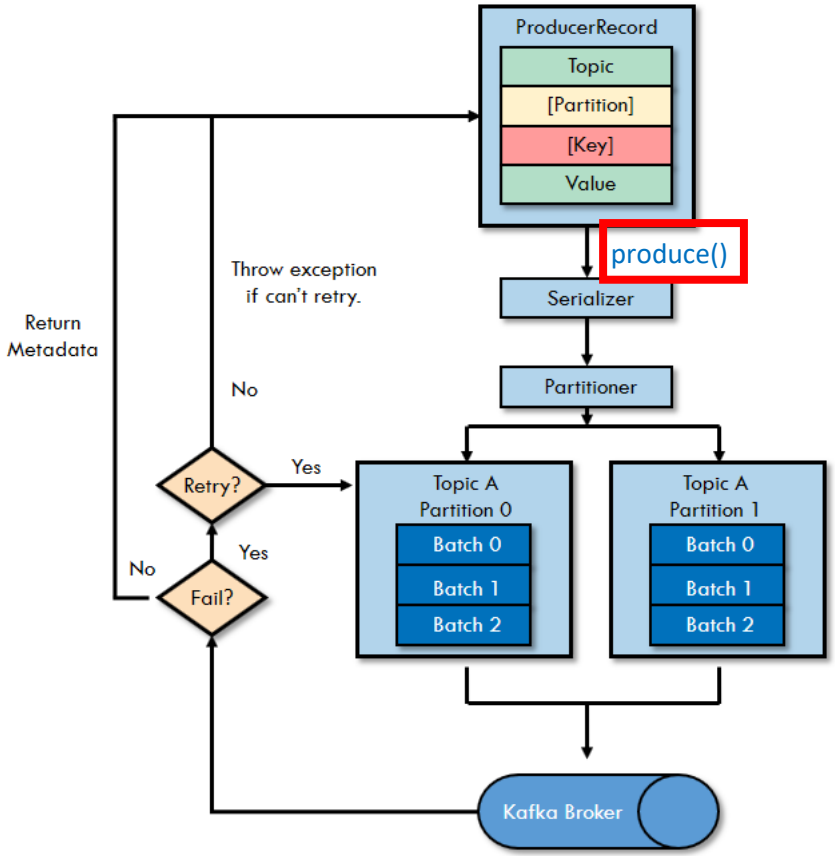
- **Fire-and-forget**
  - We send a message to the server and don't really care if it arrives successfully or not.
- ~~**Synchronous send**~~
  - ~~We send a message, and wait to see if the produce() was successful or not.~~
- **Asynchronous send**
  - We call the **produce()** method with a **callback function**, which gets triggered when it receives a response from the Kafka broker.



Three primary methods of sending messages
Fire-and-forget

- Producer_01_FireAndForget

```
# produce(topic, [value], [key], [partition], [on_delivery], [timestamp], [headers])

# ** 示範: Fire - and -forget **
# 在以下的"prouce()"過程, 我們並沒有去檢查訊息發佈的結果
# 因此這種方法的throughput最高, 但也不知道訊息是否發佈成功或失敗

for i in range(0, msgCount):
    producer.produce(topicName, key=str(i), value='msg_'+str(i))

    producer.poll(0)  # <-- (重要) 呼叫poll來讓client程式去檢查內部的Buffer
```

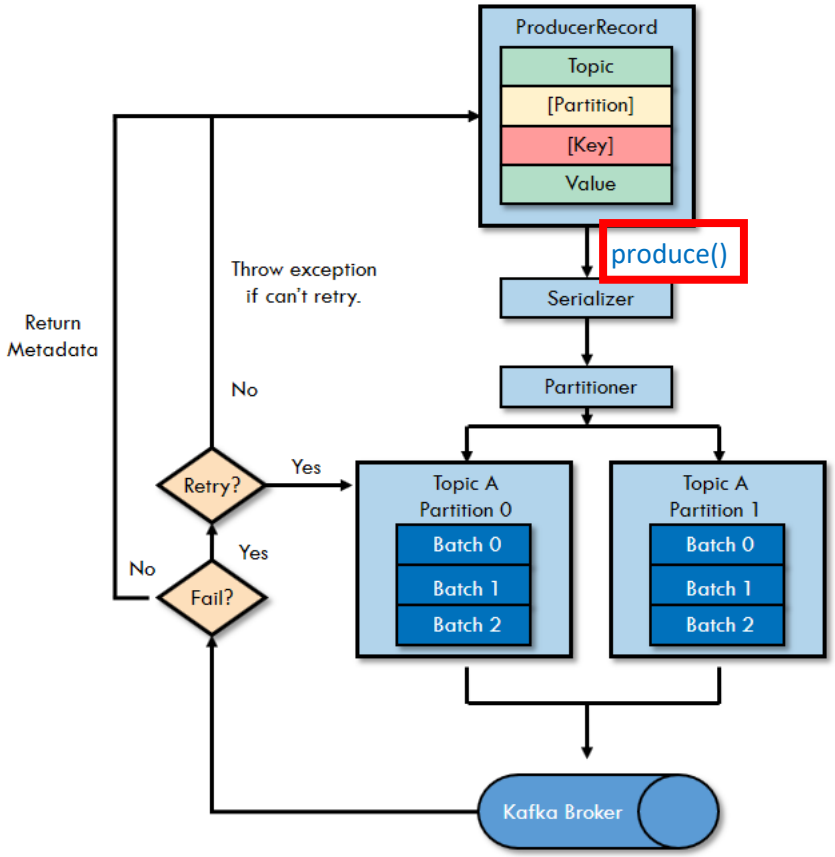


Three primary methods of sending messages
Synchronous send

- Producer_02_Sync

Python版本的Kafka client (confluent-kafka)的底層函式庫是librdkafka (用C/C++開發)。Librdkafka都是使用”異步(async)”的手法來與kafka溝通, 它不支援”同步(sync)”的發佈!!

$ pip install confluent-kafka==1.0.0rc1

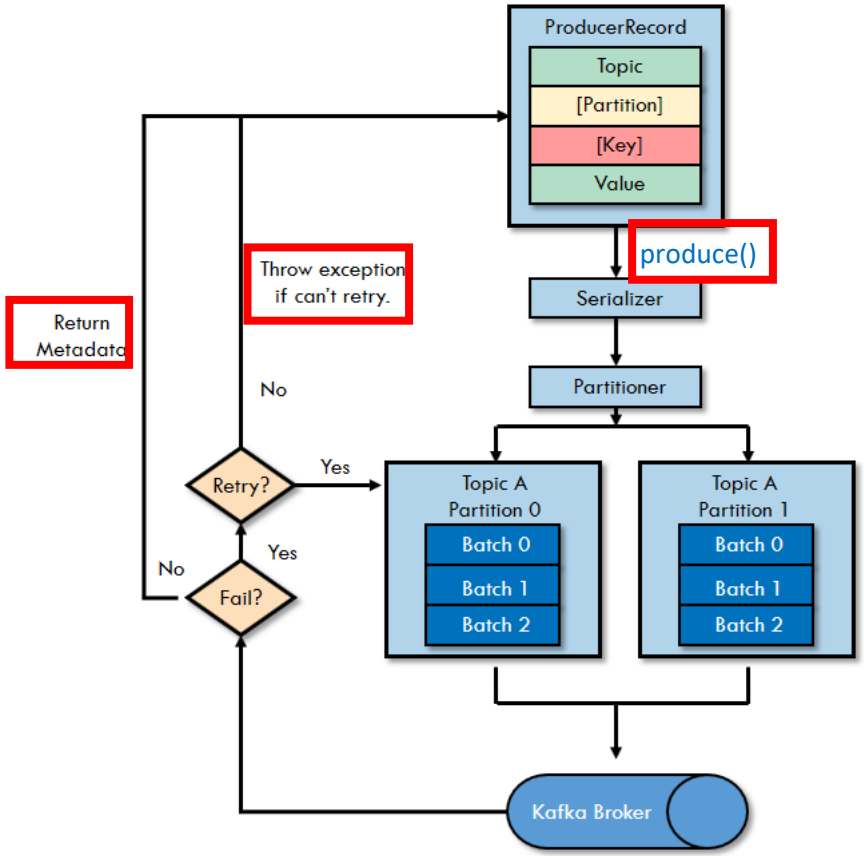


Three primary methods of sending messages
Asynchronous send

- Producer_03_Async
  - delivery_callback

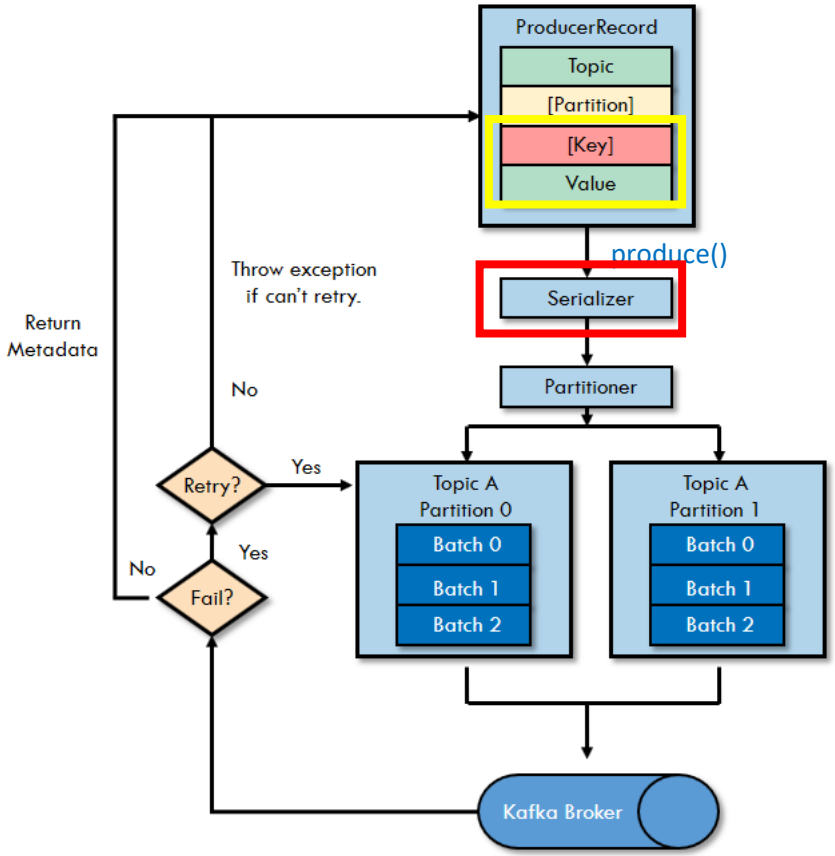
```
# produce(topic, [value], [key], [partition], [on_delivery], [timestamp], [headers])
'''
// ** 示範: Asynchronous Send **
// 透過一個Callback函式我們可以非同步地取得由Broker回覆訊息發佈的ack結果
// 這種方法可以取得Broker回覆訊息發佈的ack結果，同時又可以取得好的throughput (建議的作法)
'''

for i in range(0, msgCount):
    producer.produce(topicName, key=str(i), value='msg_'+str(i), callback=delivery_callback)

    producer.poll(0) # 呼叫poll來讓client程式去檢查內部的Buffer, 並觸發callback
```

```
# Optional per-message delivery callback (triggered by poll() or flush())
# when a message has been successfully delivered or permanently
# failed delivery (after retries).
def delivery_callback(err, msg):
    if err:
        sys.stderr.write('%% Message failed delivery: %s\n' % err)
    else:
        # 為了不讓打印訊息拖慢速度, 我們每1萬打印一筆recordMetata來看
        if int(msg.key())%10000==0:
            sys.stderr.write('%% Message delivered to topic:[%s]-partition:[%d] @ offset[%o]\n' %
                             (msg.topic(), msg.partition(), msg.offset()))
```



Configuring Producers - Serializers

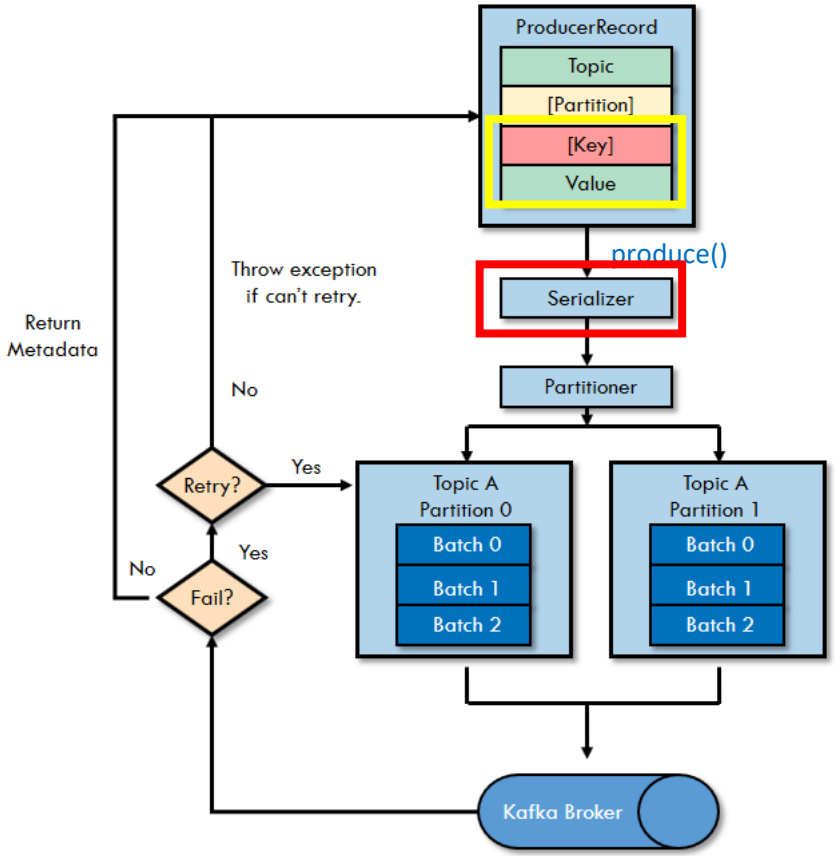
Kafka lets us publish and subscribe to streams of records and the records can be of any type (JSON, String, Object, etc.)



Configuring Producers - Serializers

- Producer_04_Json

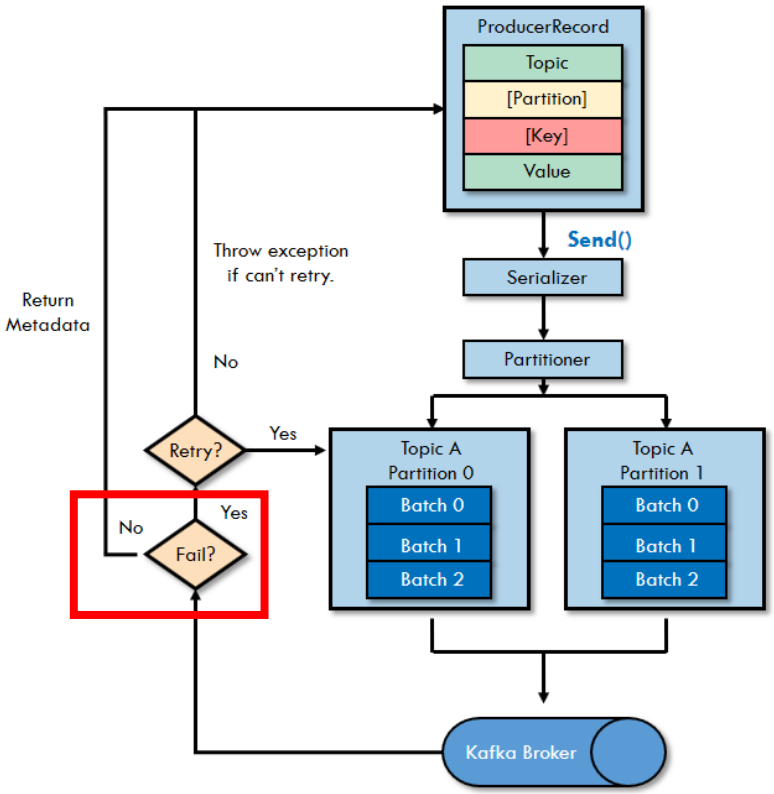
```
for i in range(0, msgCount):
    fakeNumber = str(i)
    # 讓我們產生假的Employee資料
    employee = Employee(id='empid_'+fakeNumber,
                        firstName='fn_'+fakeNumber,
                        lastName='ln_'+fakeNumber,
                        deptId='deptid_'+str(i%10),
                        hireDate=epoch_now_mills(),
                        wage=float(i),
                        sex=True
                        )

    # 轉換成JSON字串
    employeeJson = json.dumps(employee.__dict__)

    # 送出訊息
    producer.produce(topicName, key=str(i), value=employeeJson, callback=delivery_callback)

    producer.poll(0) # 呼叫poll來讓client程式去檢查內部的Buffer, 並觸發callback
```

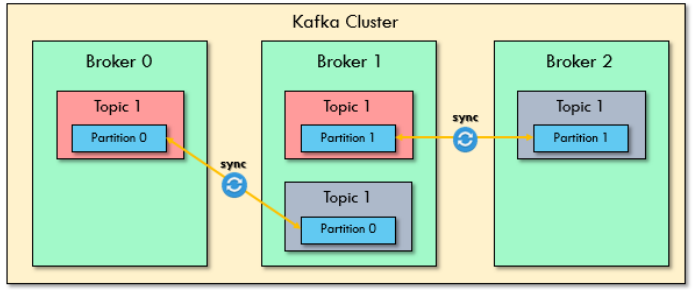


Configuring Producers

- **acks=0**, the producer will not wait for a reply from the broker before assuming the message was sent successfully.
- **acks=1**, the producer will receive a success response from the broker the moment the leader replica received the message.
- **acks=all**, the producer will receive a success response from the broker once all in-sync replicas received the message.

```
props = {
    # Kafka集群在那裡?
    'bootstrap.servers': '10.34.4.109:9092',    # <-- 置換成要連接的Kafka集群
    'acks': 0,                                  # 不等待Broker的回應
    'error_cb': error_cb                        # 設定接收error訊息的callback函數
}
```

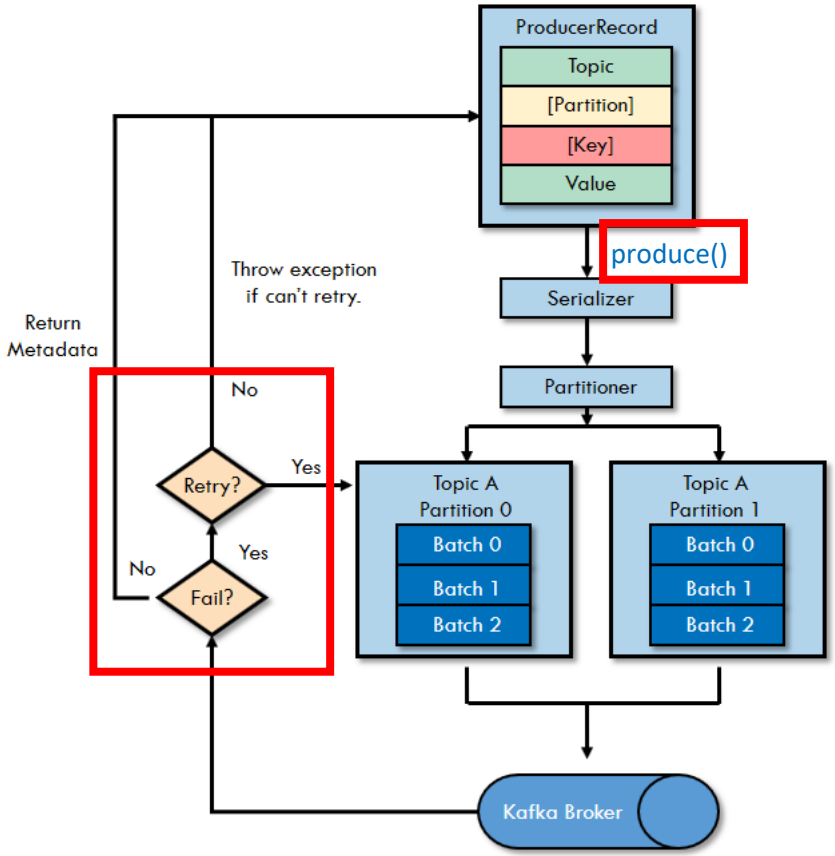




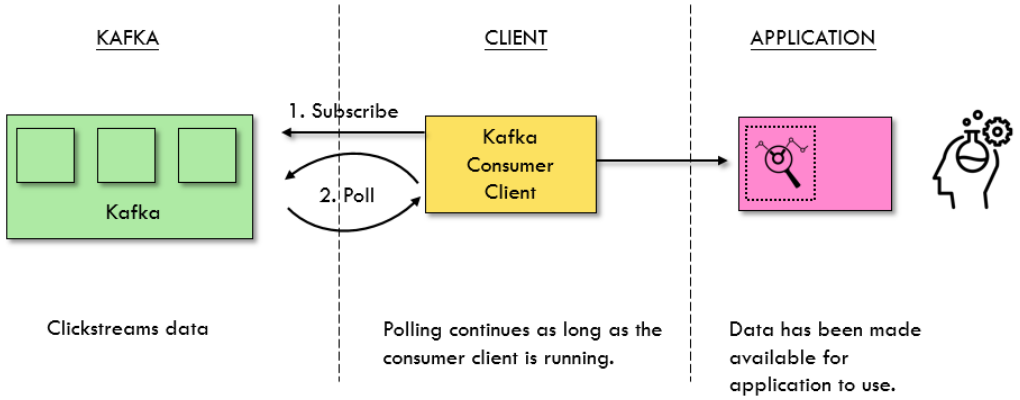
Configuring Producers

- **enable.idempotence**
  - When set to **true**, the producer will ensure that messages are successfully produced exactly once and in the original produce order.

```
props = {
    # Kafka集群在那裡?
    'bootstrap.servers': '10.34.4.109:9092',      # <-- 置換成要連接的Kafka集群
    'enable.idempotence': True,                   # 啟動idempotent producer
    'error_cb': error_cb                          # 設定接收error訊息的callback函數
}
```



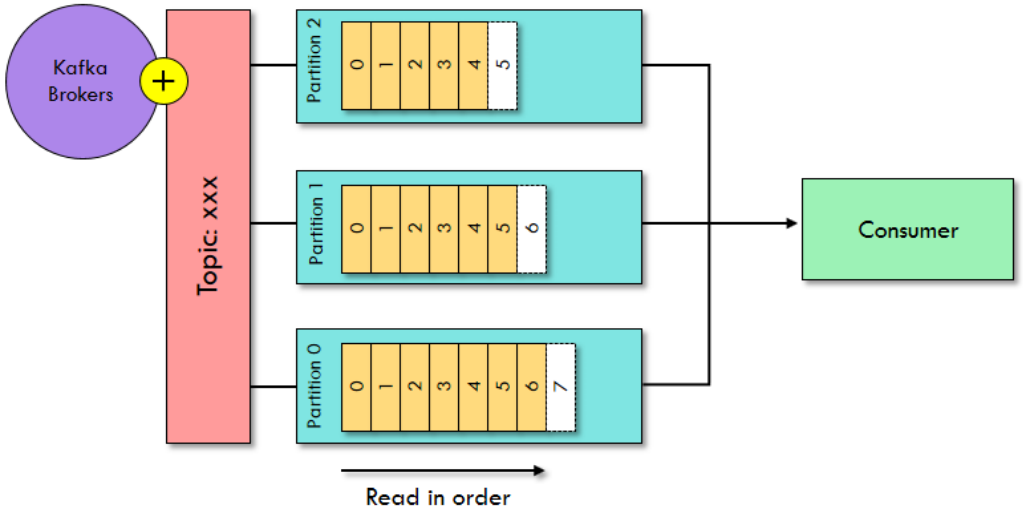
參考: https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md



Kafka Consumers: Reading Messages from Kafka

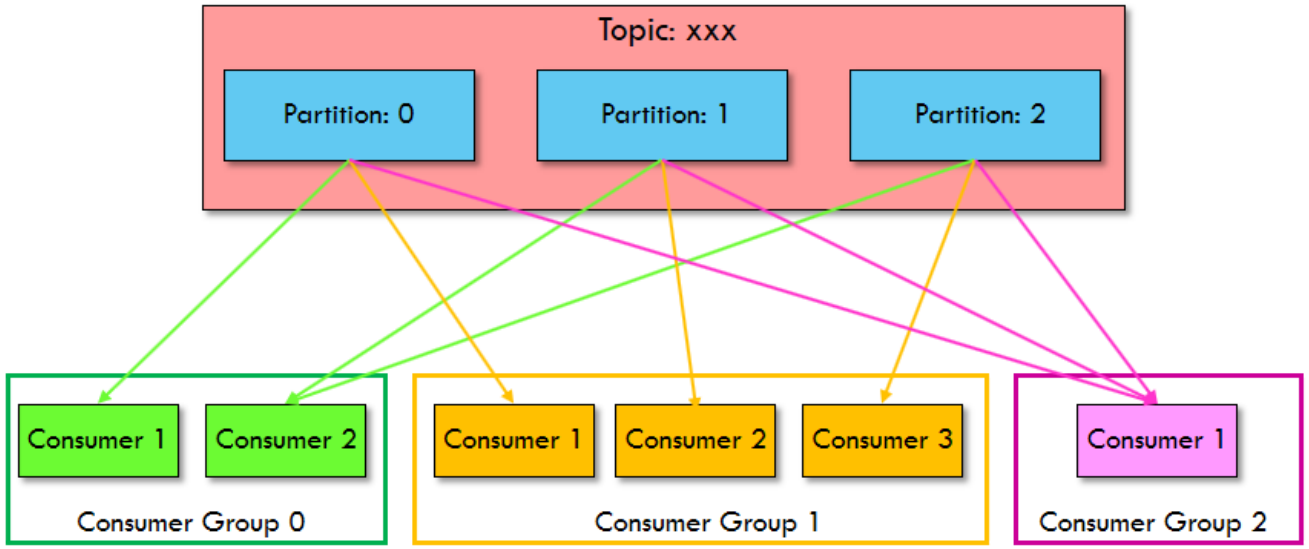
Consumers

- **Consumers** read data from a topic
- They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of pulling the data from the right brokers
- Data is read in order **for each partitions**



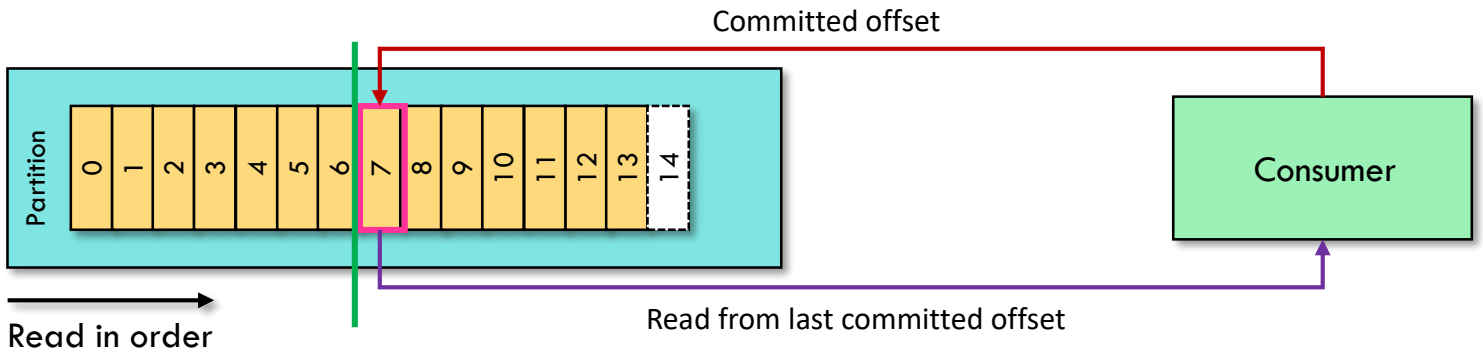
Consumer Groups

- Consumers read data in **consumer groups**
- Each consumer within a group reads from exclusive partitions
- You cannot have more consumers than partitions (otherwise some will be inactive)



Consumer Offsets

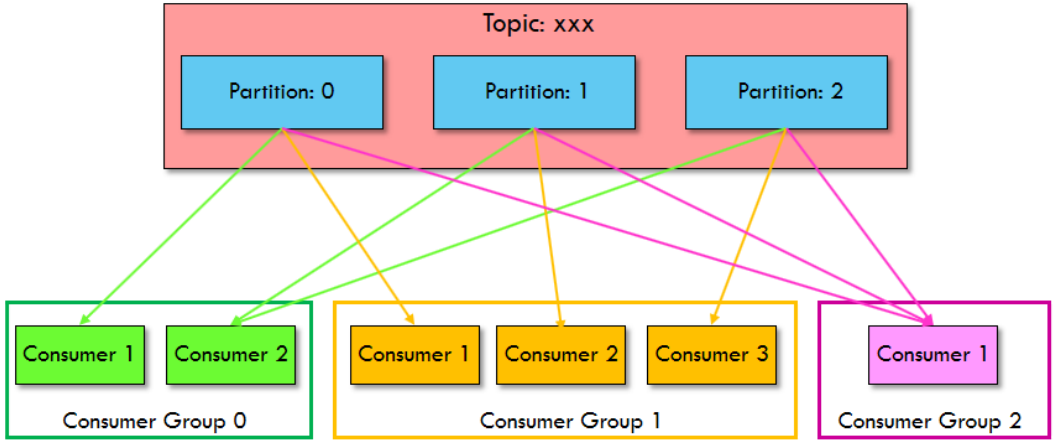
- Kafka stores the offsets at which a **consumer group** has been reading
- The **offsets** commit live in a Kafka topic named "**__consumer_offsets**"
- When a consumer has processed data received some Kafka, it should be **committing** the **offsets**
- If a consumer process dies, it will be able to read back from where it left off thanks to consumer offsets!



Configuring Consumers

- **group.id**
  - Kafka consumers are typically part of a **consumer group**. When multiple consumers are subscribed to a topic and belong to the same consumer group, each consumer in the group will receive messages from a different subset of the partitions in the topic.

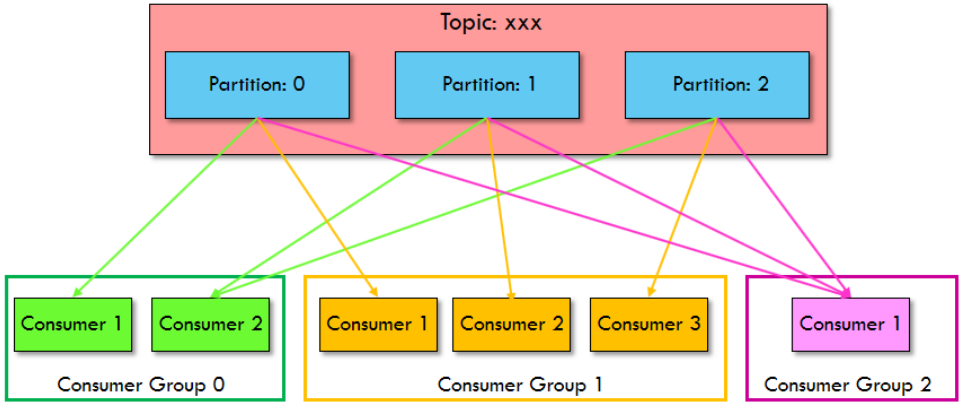
```
props = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'tdea',
    'auto.offset.reset': 'earliest',
    'session.timeout.ms': 6000,
    'error_cb': error_cb
}
```



Consumers and Consumer Groups

Create a topic with 4 partitions

```
$ kafka-topics
    --create
    --zookeeper localhost:2181
    --replication-factor 1 --partitions 4
    --topic ak03.fourpartition
```




```
root@kafka:/# kafka-topics --create --zookeeper zookeeper:2181 \
>   --replication-factor 1 \
>   --partitions 4 \
>   --topic ak03.fourpartition
WARNING: Due to limitations in metric names, topics with a period ('.') or under
score ('_') could collide. To avoid issues it is best to use either, but not bot
h.
Created topic "ak03.fourpartition".
```

在container裡頭的zookeeper服務的Hostname

Topic is created!

Consumers and Consumer Groups

Producer_05_CGroup : Publish Event:

```python
# 用來接收從Consumer instance發出的error訊息
def error_cb(err):
    print('Error: %s' % err)

# 主程式進入點
if __name__ == '__main__':
    # 步驟1. 設定要連線到Kafka集群的相關設定
    props = {
        # Kafka集群在那裡?
        'bootstrap.servers': 'localhost:9092',    # <-- 置換成要連接的Kafka集群
        'error_cb': error_cb                              # 設定接收error訊息的callback函數
    }
    # 步驟2. 產生一個Kafka的Producer的實例
    producer = Producer(**props)
    # 步驟3. 指定想要發佈訊息的topic名稱
    topicName = 'ak03.fourpartition'
    msgCount = 10000
    try:
        print('Start sending messages ...')
        # produce(topic, [value], [key], [partition], [on_delivery], [timestamp], [headers])
        for i in range(0, msgCount):
            producer.produce(topicName, key=str(i), value='msg_'+str(i))
            producer.poll(0)  # <-- (重要) 呼叫poll來讓client程式去檢查內部的Buffer
            print('key={}, value={}'.format(str(i), 'msg_'+str(i)))
            time.sleep(3) # 讓主執行緒停個3秒

        print('Send ' + str(msgCount) + ' messages to Kafka')
    except BufferError as e:
        # 錯誤處理
        sys.stderr.write('%% Local producer queue is full (%d messages awaiting delivery): try again\n' % len(producer))
    except Exception as e:
        print(e)
    # 步驟5. 確認所有在Buffer裡的訊息都已經送出去給Kafka了
    producer.flush(10)
    print('Message sending completed!')
```

這個Producer的目的是持續發佈有序列號的訊息來觀察ConsuermGroup的概念!

每3秒發佈一筆訊息 ！

Consumers and Consumer Groups

- Producer_05_CGroup

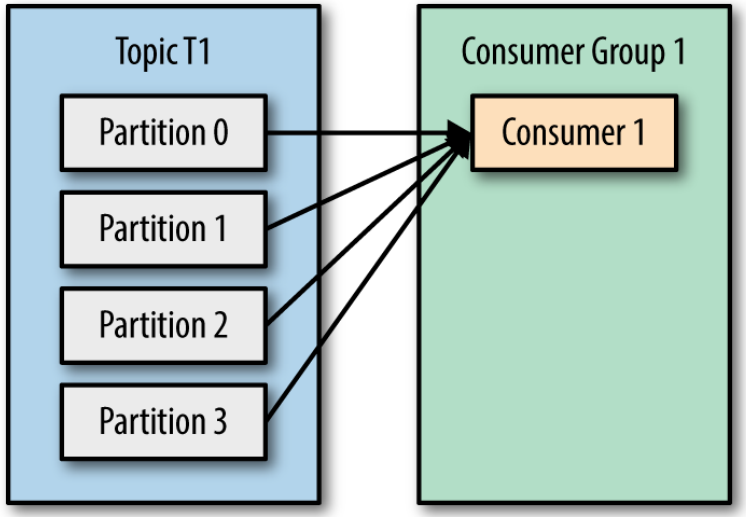
```
for i in range(0, msgCount):
    producer.produce(topicName, key=str(i), value='msg_'+str(i))
    producer.poll(0)  # <-- (重要) 呼叫poll來讓client程式去檢查內部的Buffer
    print('key={}, value={}'.format(str(i), 'msg_'+str(i)))
    time.sleep(3) # 讓主執行緒停個3秒
```

- Consumer_01_Cgroup
  - One Consumer instance

跑第1個instance, 並觀察Console的log

```
Setting newly assigned partitions: [ak03.fourpartition-0, ak03.fourpartition-1, ak03.fourpartition-2, ak03.fourpartition-3]
```

One Consumer with four partitions



Consumers and Consumer Groups

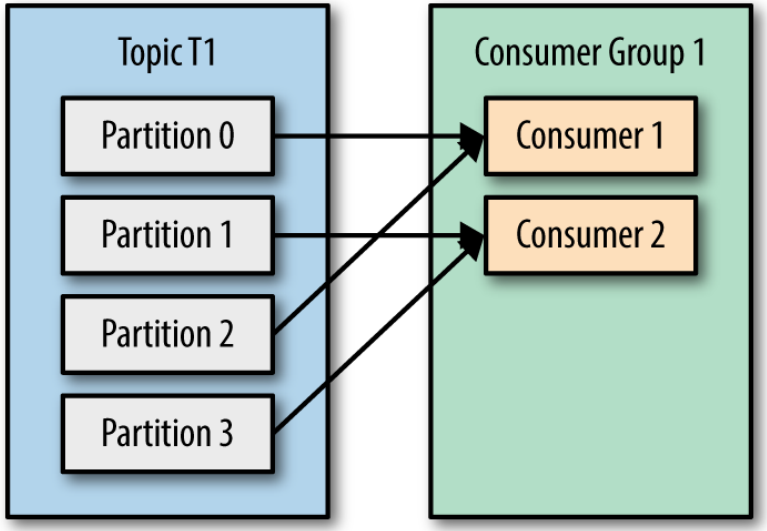
- Producer_05_CGroup

```
for(int i=0; i<msgCount; i++) {
    producer.send(new ProducerRecord<>(topicName,  key: ""+i,  value: "msg_"+i));

    Thread.sleep( millis: 3000); // 讓主執行緒停個3秒
}
```

- Consumer_01_Cgroup
  - Two Consumer instances

跑第2個instance, 並觀察Console的log

Two Consumer with four partitions




```
Revoking previously assigned partitions: [ak03.fourpartition-0, ak03.fourpartition-1, ak03.fourpartition-2, ak03.fourpartition-3]
Setting newly assigned partitions: [ak03.fourpartition-0, ak03.fourpartition-1]

Setting newly assigned partitions: [ak03.fourpartition-2, ak03.fourpartition-3]
```

Consumers and Consumer Groups

- Producer_05_CGroup

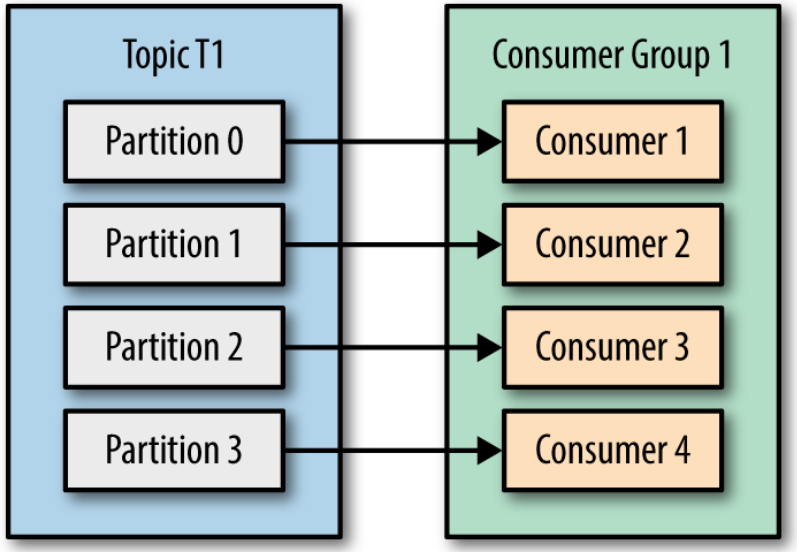
```
for(int i=0; i<msgCount; i++) {
    producer.send(new ProducerRecord<>(topicName,  key: ""+i,  value: "msg_"+i));

    Thread.sleep( millis: 3000); // 讓主執行緒停個3秒
}
```

- Consumer_01_Cgroup
  - Four Consumer instances

跑第4個instance, 並觀察Console的log

```
Setting newly assigned partitions: [ak03.fourpartition-0]
Setting newly assigned partitions: [ak03.fourpartition-3]
Setting newly assigned partitions: [ak03.fourpartition-1]
Setting newly assigned partitions: [ak03.fourpartition-2]
```

Three Consumer with four partitions



Consumers and Consumer Groups

- Producer_05_CGroup

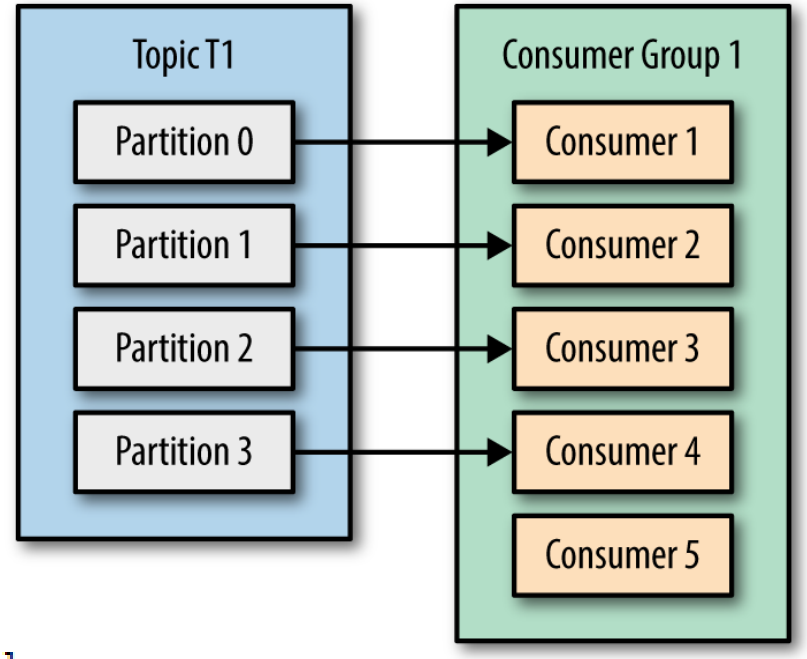
```
for(int i=0; i<msgCount; i++) {
    producer.send(new ProducerRecord<>(topicName,  key: ""+i,  value: "msg_"+i));

    Thread.sleep( millis: 3000); // 讓主執行緒停個3秒
}
```

- Consumer_01_Cgroup
  - Five Consumer instances

跑第5個instance, 並觀察Console的log

```
Setting newly assigned partitions: [ak03.fourpartition-0]
Setting newly assigned partitions: [ak03.fourpartition-3]
Setting newly assigned partitions: [ak03.fourpartition-1]
Setting newly assigned partitions: [ak03.fourpartition-2]
Setting newly assigned partitions: []
```

Five Consumer with four partitions



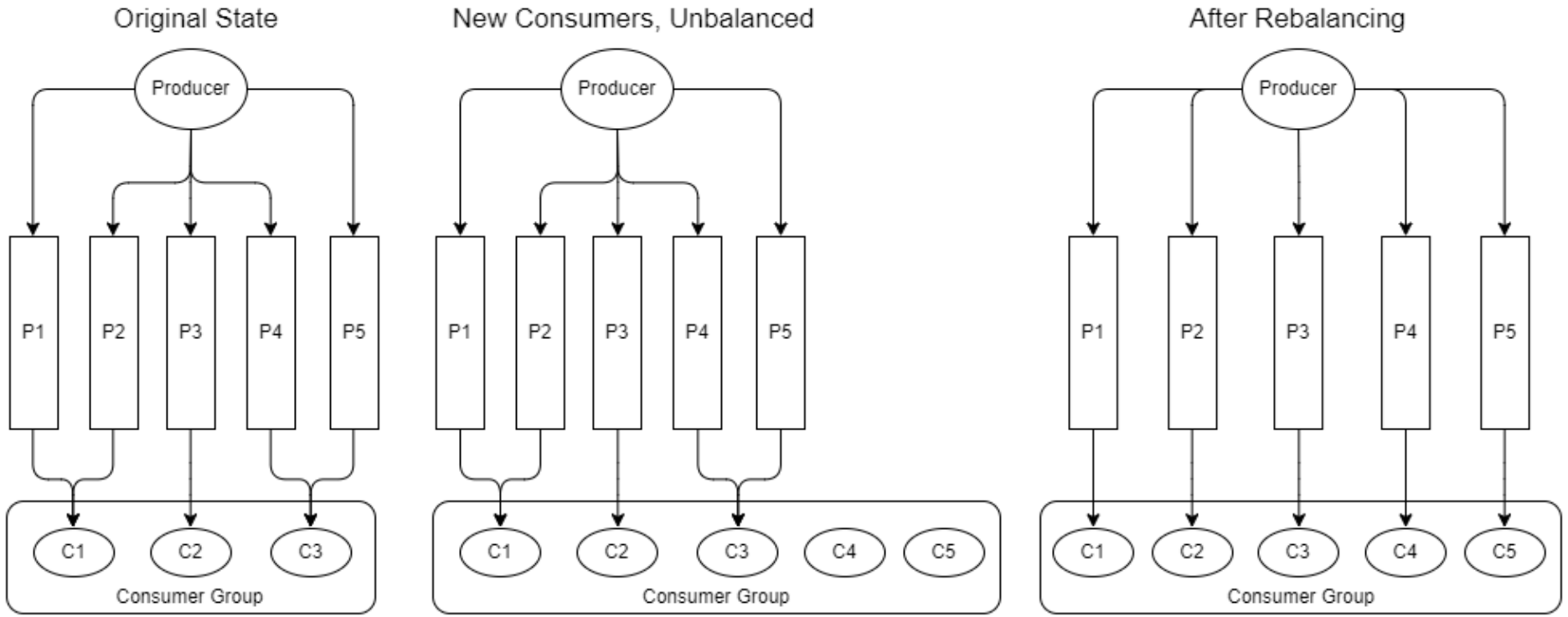
What is Rebalance?

- Every consumer in a **consumer group** is assigned one or more topic partitions exclusively, and **Rebalance** is the re-assignment of partition ownership among consumers.
- A Rebalance happens when:
  - a consumer JOINS the group
  - a consumer SHUTS DOWN cleanly
  - a consumer is considered DEAD by the group coordinator. This may happen after a crash or when the consumer is busy with a long-running processing
  - new partitions are added

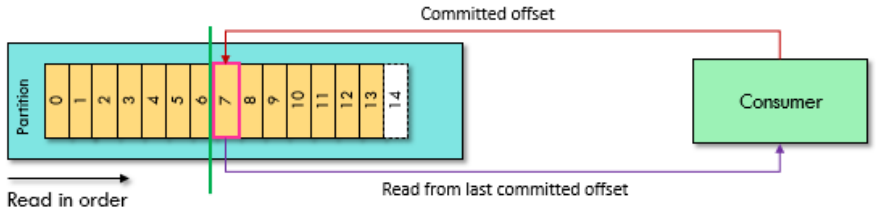
Rebalance process

- **Rebalance** can be more or less described as follows:
  - The leader receives a list of all consumers in the group from the group coordinator and is responsible for assigning a subset of partitions to each consumer.
  - After deciding on the partition assignment the group leader sends the list of assignments to the group coordinator, which sends this information to all the consumers.

Rebalance process



Commits and Offsets



- Whenever we call **consume()**, broker returns records that consumers in our group have not read yet.
- This means that Kafka client tracking which records were read by a consumer of the group.
- We call the action of updating the current position in the partition a **commit**.

```
while True:
    # 請求Kafka把新的訊息吐出來
    records = consumer.consume(num_messages=500, timeout=1.0)  # 批次讀取
```

Commits and Offsets

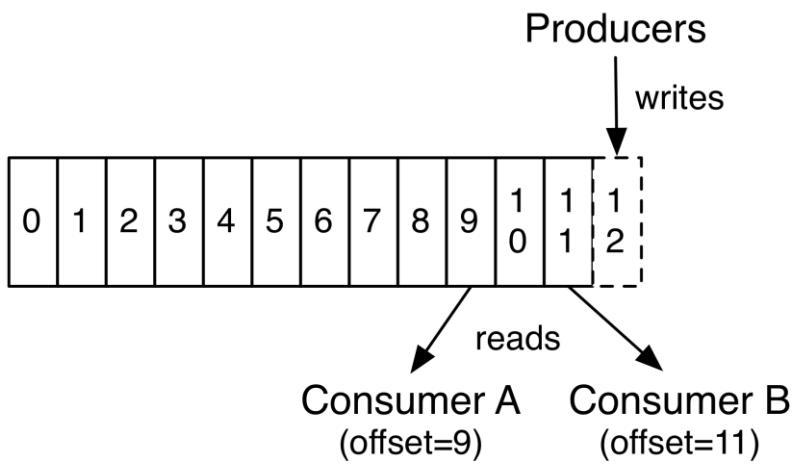
- How does a consumer commit an offset?
- Kafka client produces a message to Kafka, to a special **__consumer_offsets** topic, with the committed offset for each partition.
- If a consumer crashes or a new consumer joins the consumer group, it will trigger a **partition rebalance**.
- After a **rebalance**, each consumer may be assigned a new set of partitions than the one it processed before.
- The consumer will read the latest committed offset of each partition and continue from there.

Commits and Offsets

- Managing offsets has a big impact on the client application.
- The Kafka Consumer API provides multiple ways of committing offsets:
  - Automatic Commit
  - Synchronous Commit (previous **consume()** offset)
  - Asynchronous Commit (previous **consume()** offset)
  - ~~Combining Synchronous and Asynchronous Commits (previous **consume()** offset)~~
  - Commit Specified Offset

Configuring Consumers

- **auto.offset.reset**
  - This property controls the behavior of the consumer when it starts reading a partition for which it doesn't have a committed offset.
  - The default is "**latest**," which means that lacking a valid offset, the consumer will start reading from the newest records
  - The alternative is "**earliest**," which means that lacking a valid offset, the consumer will read all the data in the partition, starting from the very beginning.



Commits and Offsets: Automatic Commit

- Configure below configurations:
  - **enable.auto.commit=true**
  - **auto.commit.interval.ms=5000 (default)**
- Every **five** seconds the consumer will commit the largest offset your client received from consume().
- Whenever you consume(), the consumer checks if it is time to commit, and if it is, it will commit the offsets it returned in the last consume().

Commits and Offsets: **Automatic Commit**

- Consumer_02_AutoCommit

```
props = {
    'bootstrap.servers': 'localhost:9092',        # Kafka集群在那裡? (置換成要連接的Kafka集群)
    'group.id': 'tdea',                             # ConsumerGroup的名稱 (置換成你/妳的學員ID)
    'auto.offset.reset': 'earliest',                # 是否從這個ConsumerGroup尚未讀取的partition/offset開始讀
    'enable.auto.commit': True,                     # 是否啟動自動commit
    'auto.commit.interval.ms': 5000,                # 自動commit的interval
    'on_commit': print_commit_result,               # 設定接收commit訊息的callback函數
    'error_cb': error_cb                            # 設定接收error訊息的callback函數
}
```

```
ak03.test-0-50 : (41 , msg_41)
ak03.test-0-51 : (42 , msg_42)
# Committed offsets for: ak03.test-0 {offset=53}
ak03.test-0-52 : (43 , msg_43)
ak03.test-0-53 : (44 , msg_44)
# Committed offsets for: ak03.test-0 {offset=54}
ak03.test-0-54 : (45 , msg_45)
ak03.test-0-55 : (46 , msg_46)
# Committed offsets for: ak03.test-0 {offset=56}
ak03.test-0-56 : (47 , msg_47)
```

Commits and Offsets: Automatic Commit

- With **auto-commit** enabled, a call to consume() *will always commit the last offset returned by the previous poll*. It doesn't know which events were actually processed, so it is critical to always process all the events returned by consume() before calling consume() again.
- Automatic commits are convenient, but they don't give developers enough control to avoid duplicate messages.

Commits and Offsets: Manual Commit

- The consumer API has the option of committing the current offset at a point that makes sense to the application developer rather than based on a timer.
- Configure below configurations:
  - **auto.commit.offset=false**
- Offsets will only be committed when the application explicitly chooses to do so.
- The simplest and most reliable of the commit APIs is：
  - **consumer.commit(asynchronous=False)**

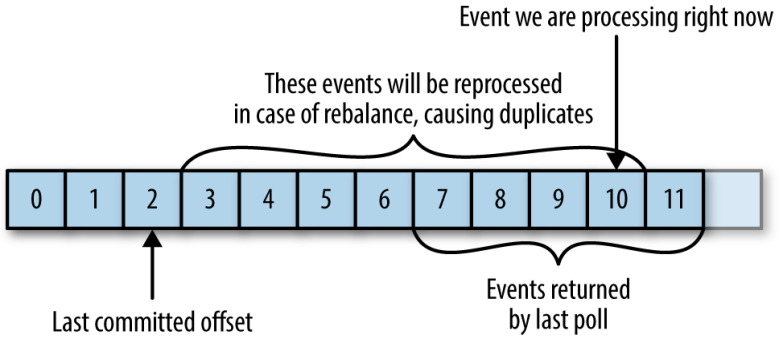
Commits and Offsets: Sync Commit

- Consumer_03_CommitSync

```
props = {
    'bootstrap.servers': 'localhost:9092',        # Kafka集群在那裡? (置換成要連接的Kafka集群)
    'group.id': 'tdea',                             # ConsumerGroup的名稱 (置換成你/妳的學員ID)
    'auto.offset.reset': 'earliest',                # 是否從這個ConsumerGroup尚未讀取的partition/offset開始讀
    'enable.auto.commit': False,                    # 是否啟動自動commit
    'error_cb': error_cb                            # 設定接收error訊息的callback函數
}

# 打印SyncCommit的結果
def print_sync_commit_result(partitions):
    if partitions is None:
        print('# Failed to commit offsets')
    else:
        for p in partitions:
            print('# Committed offsets for: %s-%s {offset=%s}' % (p.topic, p.partition, p.offset))

# 同步地執行commit (Sync commit)
if(recrods_pulled):
    offsets = consumer.commit(asynchronous=False)
    print_sync_commit_result(offsets)
```

Commits and Offsets: Sync Commit

- Use **commit()** without parameter will commit the latest offset returned by **consume()** and return once the offset is committed, throwing an exception if commit fails for some reason.
- Make sure you call **commit()** after you are done processing all the records in the collection.
- When **rebalance** is triggered, all the messages from the beginning of the most recent batch until the time of the rebalance will be processed twice.



Commits and Offsets: Asynchronous Commit

- One drawback of manual commit is that the application is ***blocked*** until the broker responds to the commit request.
- Another option is the **asynchronous** commit. Instead of waiting for the broker to respond to a commit, we just send the request and continue on.

Commits and Offsets: Asynchronous Commit

- Consumer_04_CommitAsync

```
props = {
    'bootstrap.servers': 'localhost:9092',        # Kafka集群在那裡? (置換成要連接的Kafka集群)
    'group.id': 'tdea',                             # ConsumerGroup的名稱 (置換成你/妳的學員ID)
    'auto.offset.reset': 'earliest',                # 是否從這個ConsumerGroup尚未讀取的partition/offset開始讀
    'enable.auto.commit': False,                    # 是否啟動自動commit
    'on_commit': print_commit_result,               # 設定接收commit訊息的callback函數
    'error_cb': error_cb                            # 設定接收error訊息的callback函數
}
```

```
        recrods_pulled = True
        # ** 在這裡進行商業邏輯與訊息處理 **
        # 取出相關的metadata
        topic = record.topic()
        partition = record.partition()
        offset = record.offset()
        timestamp = record.timestamp()
        # 取出msgKey與msgValue
        msgKey = try_decode_utf8(record.key())
        msgValue = try_decode_utf8(record.value())

        # 秀出metadata與msgKey & msgValue訊息
        print('%s-%d-%d : (%s , %s)' % (topic, partition, offset, msgKey, msgValue))
    # 異步地執行commit (Async commit)
    if(recrods_pulled):
        consumer.commit()
```

Commits and Offsets: Asynchronous Commit

- **consumer.commit()** gives you an option to pass in a callback that will be triggered when the broker responds.

```
props = {
    'bootstrap.servers': 'localhost:9092',        # Kafka集群在那裡? (置換成要連接的Kafka集群)
    'group.id': 'tdea',                             # ConsumerGroup的名稱 (置換成你/妳的學員ID)
    'auto.offset.reset': 'earliest',                # 是否從這個ConsumerGroup尚未讀取的partition/offset開始讀
    'enable.auto.commit': False,                    # 是否啟動自動commit
    'on_commit': print_commit_result,               # 設定接收commit訊息的callback函數
    'error_cb': error_cb                            # 設定接收error訊息的callback函數
}
```

```
# 當發生commit時被呼叫
def print_commit_result(err, partitions):
    if err is not None:
        print('# Failed to commit offsets: %s: %s' % (err, partitions))
    else:
        for p in partitions:
            print('# Committed offsets for: %s-%s {offset=%s}' % (p.topic, p.partition, p.offset))
```

Commits and Offsets: Combining Sync and Async Commits

因為Python與Java版本 的Kafka client 在 實作上的差異, 因此本頁的內容不適合Python的client!

Commits and Offsets: Combining Sync and Async Commits

- Consumer_05_CommitSyncAsync

因為Python與Java版本 的Kafka client 在 實作上的差異, 因此本頁的內容不適合Python的client!

Commits and Offsets: Commit Specified Offset

- Committing the latest offset only allows you to commit as often as you finish processing batches.
- What if **consume()** returns a huge batch and you want to commit offsets in the middle of the batch to avoid having to process all those rows again if a **rebalance** occurs?
- You can't just call **commit (asynchronous=False)** or **commit ()** —this will commit the last offset returned, which you didn't get to process yet.
- Consumer API allows you to call **commit()** and pass a map of **partitions** and **offsets** **or** **message** that you wish to commit.

Commits and Offsets: **Commit Specified Offset**

commit([*message=None*] [, *offsets=None*] [, *asynchronous=True*])

Commit a message or a list of offsets.

`message` and `offsets` are mutually exclusive, if neither is set the current partition assignment's offsets are used instead. The consumer relies on your use of this method if you have set 'enable.auto.commit' to False

Parameters:

- **message** (*confluent_kafka.Message*) – Commit message's offset+1.
- **offsets** (*list(TopicPartition)*) – List of topic+partitions+offsets to commit.
- **asynchronous** (*bool*) – Asynchronous commit, return None immediately. If False the commit() call will block until the commit succeeds or fails and the committed offsets will be returned (on success). Note that specific partitions may have failed and the .err field of each partition will need to be checked for success.

Commits and Offsets: Commit Specified Offset

- Consumer_06_CommitSpecified

```
props = {
    'bootstrap.servers': 'localhost:9092',        # Kafka集群在那裡? (置換成要連接的Kafka集群)
    'group.id': 'tdea',                             # ConsumerGroup的名稱 (置換成你/妳的學員ID)
    'auto.offset.reset': 'earliest',                # 是否從這個ConsumerGroup尚未讀取的partition/offset開始讀
    'enable.auto.commit': False,                    # 是否啟動自動commit
    'on_commit': print_commit_result,               # 設定接收commit訊息的callback函數
    'error_cb': error_cb                            # 設定接收error訊息的callback函數
}
```

```
# ** 在這裡進行商業邏輯與訊息處理 **
# 取出相關的metadata
topic = record.topic()
partition = record.partition()
offset = record.offset()
timestamp = record.timestamp()
# 取出msgKey與msgValue
msgKey = try_decode_utf8(record.key())
msgValue = try_decode_utf8(record.value())

# 秀出metadata與msgKey & msgValue訊息
print('%s-%d-%d : (%s , %s)' % (topic, partition, offset, msgKey, msgValue))

consumer.commit(record) # 非同步的commit
```

Rebalance callback

- If you know your consumer is about to lose ownership of a partition, you will want to commit offsets of the last event you've processed.
- The consumer API allows you to run your own code when partitions are added or removed from the consumer.
- You do this by passing a **callback** when calling the subscribe() method

```
consumer.subscribe([topicName], on_assign=print_assignment, on_revoke=commit_on_revoke)
```

Commits and Offsets: RebalanceCallback

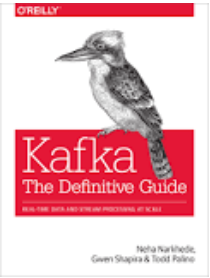
- Consumer_07_CommitSpecified

```python
props = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'tdea',
    'auto.offset.reset': 'earliest',
    'enable.auto.commit': False,
    'on_commit': print_commit_result,
    'error_cb': error_cb
}


# 步驟2. 產生一個Kafka的Consumer的實例
consumer = Consumer(props)
# 步驟3. 指定想要訂閱訊息的topic名稱
topicName = 'ak03.fourpartition'
# 步驟4. 讓Consumer向Kafka集群訂閱指定的topic
consumer.subscribe([topicName]
                   , on_assign=print_assignment
                   , on_revoke=commit_on_revoke)
```

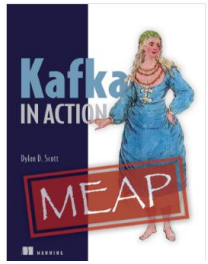
```python
# 當Rebalance被觸發後, Commit現在process的offets
def commit_on_revoke(consumer, partitions):
    global offsets_dict

    result = '['
    is_first = True
    for p in partitions:
        if is_first:
            result = result + '{}-{}'.format(p.topic, p.partition)
            is_first = False
        else:
            result = result + ', {}-{}'.format(p.topic, p.partition)
    result = result + ']'
    print('Revoking previously assigned partitions: ' + result)

    for k, v in offsets_dict.items():
        consumer.commit(v) # 進行異步的commit
```

Reference

1. Kafka: The Definitive Guide – O'REILLY



2. Kafka In Action - MANNING



3. Learn Apache Kafka for Beginners – Udemy (Stephane Maarek)

4. Confluent Document of Kafka – confluent.io

confluent


The
End