**Configuración**

**Procesamiento de eventos concurrente**

Indicaciones para el micro :   
<https://github.com/santander-group-spain-gln/san-coplst-plardpcaslate/commits/feature/thread-pool-refactor/>

Configuración de ejemplo para **PRE/PRO**:

darwin:  
 core:  
    async:  
      enabled: true  
      core-pool-size: 20  
      max-pool-size: 20  
      queue-capacity: 10  
      thread-name-prefix: EventProcessingThread-  
      thread-pool-policy: CALLER\_RUNS\_POLICY

Recomiendo tener el mismo valor en max-pool-size y core-pool-size a no ser que se necesite aumentar mucho el máximo de hilos.

En **Local** he dejado:

darwin:  
 core:  
    async:  
      enabled: true  
      core-pool-size: 1  
      max-pool-size: 1  
      queue-capacity: 1  
      thread-name-prefix: EventProcessingThread-  
      thread-pool-policy: CALLER\_RUNS\_POLICY

Para poder probar qué pasa cuando llega un evento y no hay ningún hilo disponible para procesarlo, ni espacio en la cola. CALLER\_RUNS\_POLICY hace que el hilo que recibe el evento lo procese él mismo, en vez de rechazarlo. Al mantener ocupado el hilo que recibe el evento, se evita que se rechace el procesamiento de los siguientes eventos que lleguen.

**Reintentos de lectura de la cola MQ**

Para configurar los reintentos de lectura de la cola MQ se puede elegir el intervalo entre reintentos y el número máximo de reintentos:

retry:  
 max-attempts: ${env.retry.max-attempts}  
 interval: ${env.retry.interval}

**Pruebas**

Se puede probar con la herramienta kafka-avro-producer (<https://santandernet.sharepoint.com/sites/tech-platform-spain/SitePages/Kafka-Avro-Producer-Tool.aspx>).

Por ejemplo, para probar qué pasa cuando llega un evento y no hay ningún hilo disponible para procesarlo, ni espacio en la cola, se pueden mandar 3 eventos:

* Uno debería ser procesado por el hilo del pool (se debería llamar EventProcessingThread-1).
* El siguiente debería ser introducido en la cola y, por tanto quedar a la espera de la liberación de EventProcessingThread-1. Tras ello, debería ser procesado por EventProcessingThread-1.
* El último debería ser procesado por el hilo que recibe el evento, ya que no hay ningún hilo disponible para procesarlo y tampoco espacio en la cola.

Para lanzar la prueba, en tres consolas diferentes, se ejecuta el siguiente comando:

java -jar .\kafka-avro-producer-1.0.2.jar -f "events\SANES.COPLST.AUTHORIZATION.DELAYED\SANES.COPLST.AUTHORIZATION.DELAYED-EXAMPLE.json" -c "kafka-config-dev.properties" -t "SANES.COPLST.AUTHORIZATION.DELAYED"

Ejemplo SANES.COPLST.AUTHORIZATION.DELAYED-EXAMPLE.json:

{"idMessage":"1","idQueue":"PCASON1","arrivalTime":"2025-08-14T14:22:59.947","typeCode":"MP0C","content":"0111201654890142000001230002000000000002790000000000000000000002792024091904335500000000242634250000014019535040000380M103001M00010000000426397800000000    000000000000000000000057750000000000740851660368806987115000102438      0423 - COOP MURANO       VENEZIA    VEIT          00157020131365163980160867011      2602   113700005260225490000001000000000000000 7851                                                                            978978978                                        00000000        ","isEncrypted":false,"creationDateTime":"2024-09-19T10:08:01.574729","cardTransaction":{"cardTransactionId":"","transactionCode":"","traceReference":""}}

**TODO**

* Arreglar los test unitarios.
* Eliminar PlardPcasLateController, innecesario ya que se pueden generar eventos con kafka-avro-producer.
* Probar correctamente. En mis pruebas siempre he obtenido errores de autenticación al conectar con las colas MQ.
* Manejar excepciones en PlardPcasLateCommonServiceImpl.receiveMessage.
* Reintentar llamadas a Stratos en caso de error PlardPcasLateCommonServiceImpl.callLateResponseService. <https://resilience4j.readme.io/docs/retry>

Retry

Getting started with resilience4j-retry

la próxima semana y la siguiente estaré de vacaciones así que échale un ojo y cualquier cosa me dices