Qualcomm Car-to-Cloud Platform

Kinesis Wrapper Usage Document

Version No.4

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Jean Johnson		
Role	Developer		
Signature			
Date	August 27, 2021		

Table of Contents

1 Project Overview

We have developed an interface adapter for access to AWS Kinesis Stream, so that applications, can consume it for publishing and subscribing message using stream.

Stream Interface

1. Publish single message to Stream
2. Publish list of messages to Stream
3. Subscribe to a Stream and Forward to processor

Processor Interface

1. Process messages received from Stream

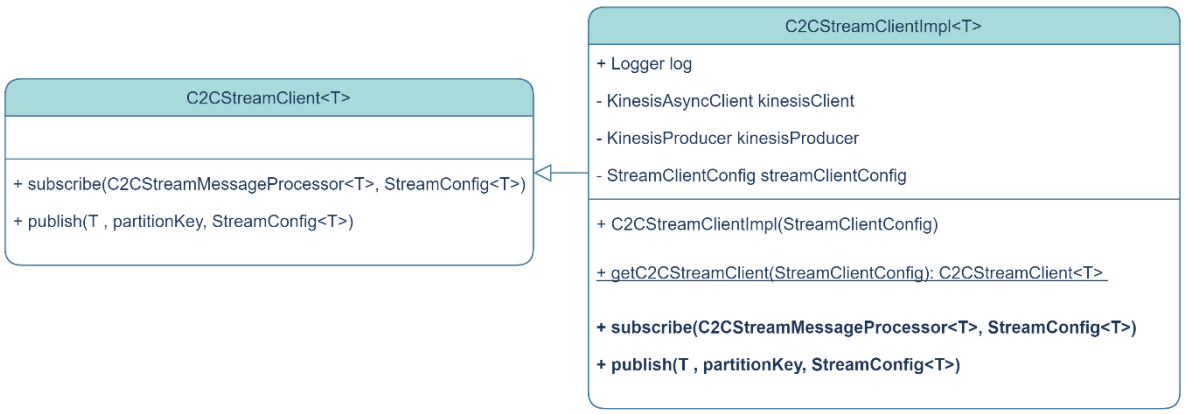
Retry Mechanism

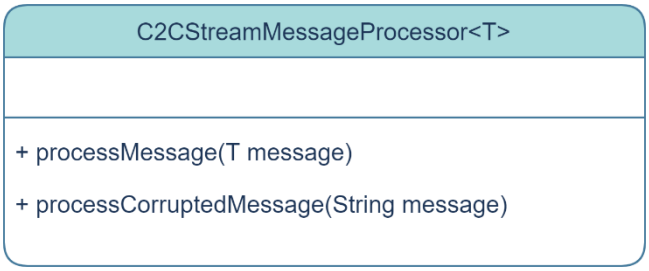
The Wrapper also executes Retry using Resilence4J Library. The *Number of attempt* and the *Interval* between the Attempt can be configured by the Client.

Currently, retry is done for the following Error Codes:

1. Provisioned Throughput Exceeded Exception
2. Kinesis Exception
3. Service Unavailable
4. Request Expired
5. Internal Failure
6. Unable to Connect to Endpoint

2 Class Diagram





3 Configurations

Class	Name	Default
Stream Client Config		
Region	Region of the stream that the Client will publish/subscribe	
streamPublisherConfig	Configuration related to publishing messages to Stream	
streamSubscriberConfig	Configuration related to subscribing messages from Stream	
Stream Publisher Config		
publisherMaxConnections	Maximum number of connections to open to the back end. HTTP requests are sent in parallel over multiple connections.	5
publisherMaxBufferTime	Maximum amount of time (milliseconds) a message may spend being buffered before it gets sent.	100
publisherMaxRecords	Maximum number of items to pack into an Put Records request.	500
publisherRequestTimeout	The maximum total time (milliseconds) elapsed between when we begin a HTTP request and receiving all of the response. If it goes over, the request will be timed-out.	1000
publisherConnectTimeout	Timeout (milliseconds) for establishing TLS connections.	6000
publishRecordTTL	Set a time-to-live on records (milliseconds). Records that do not get successfully put within the limit are	30000

	failed.	
publisherRetryMaxAttempt	The maximum number of retry attempts for a failed publish operation	3
publisherRetryDelaySeconds	The interval between each retry in seconds	3
Stream Subscriber Config		
checkpointRecordLimit	Checkpoint Record Limit: Wrapper will checkpoint after the specified limit To Force check pointing every batch, set value to -1	1000
initialLeaseTableReadCapacity	Read capacity to be provisioned for the Lease Table	30
initialLeaseTableWriteCapacity	Write capacity to be provisioned for the Lease Table.	30
coreThreadPoolSize	The size of the core thread pool	30
maxThreadPoolSize	The maximum number of threads to allow in the pool	60
threadKeepAlive	Maximum time Idle Threads will wait for new tasks before terminating (If no. of threads is greater than Core Pool Size)	5

4 Interface Method Details

Method Name	Purpose	Input	Output
C2C Stream Client	Interface for sending and receiving messages from Stream		
Subscribe (C2CStreamMessage Processor, StreamConfig)	Method to subscribe to Stream so that we can stream messages from the Stream.	1. C2CStreamMessagePr ocessor *(to process the messages)* 2. Stream Config *(configurations related to publish/subscribe)*	
publish (T, String, StreamConfig)	Method to send a single message to Stream	1. Message to publish 2. Partition key 3. Stream Config *(configurations related to publish/subscribe)*	StreamPublish Result

Method Name	Purpose	Input	Output
C2C Stream Message Processor	Interface to process messages received on Subscription to Stream		
processMessage(T)	Method to process data received from Stream	Message received	
processCorruptedMessage (String)	Method to process data received from Stream, but failed POJO Conversion to T	Invalid Json Data	

5 Usage Details

5.1 Dependencies

Stream Interface

```
<dependency>
	<groupId>com.c2c</groupId>
	<artifactId>c2c_base_stream_intf</artifactId>
	<version>1.1.1-SNAPSHOT</version>
</dependency>
```

Stream Kinesis implementation

```
<dependency>
	<groupId>com.c2c</groupId>
	<artifactId>c2c_base_stream_kinesis_impl</artifactId>
	<version>1.1.1-SNAPSHOT</version>
</dependency>
```

5.2 Stream Publisher Sample Code

```
public class SampleStreamPublisher {

  private static Logger log = LoggerFactory
          .getLogger(SampleStreamPublisher.class);

  public static void main(String[] args) {

    // CREATE STREAM CONNECTION CONFIGURATION
    StreamClientConfig streamConnectionConfig = createStreamConnectionConfig();
    streamConnectionConfig
            .setStreamPublisherConfig(new StreamPublisherConfig());
    streamConnectionConfig
            .setStreamSubscriberConfig(new StreamSubscriberConfig());
    log.info("Stream Connection Config: {}", streamConnectionConfig);

    // CREATE STREAM CONFIGURATION
    StreamConfig<CommunicationCoreMessage> streamConfig = createStreamConfig();
    log.info("Stream Connection Config: {}", streamConfig);

    // CREATE STREAM CLIENT
    C2CStreamClient<CommunicationCoreMessage> streamClient = C2CStreamClientImpl
            .getC2CStreamClient(streamConnectionConfig);
    log.info("Stream Client created");

    // CREATE A MESSAGE
    CommunicationCoreMessage c2cMessage = createMessage();
    c2cMessage.setDeviceId("SINGLE-MESSAGE-TEST");

    // PUBLISH MESSAGE
    streamClient.publish(c2cMessage, c2cMessage.getDeviceId(), streamConfig);

  }

  private static StreamClientConfig createStreamConnectionConfig() {
    String region = "<region>";

    return new StreamClientConfig(region);
  }

  private static StreamConfig<CommunicationCoreMessage> createStreamConfig() {

    StreamConfig<CommunicationCoreMessage> config = new StreamConfig<>(
            "<region>", CommunicationCoreMessage.class);

    return config;
  }

  private static CommunicationCoreMessage createMessage() {

    Map<String, Object> propertyBag = new HashMap<>();
    propertyBag.put("<property-name>", "<property-value>");

    CommunicationCoreMessage c2cMessage = new CommunicationCoreMessage();
    c2cMessage.setMessageId("<message-id>");
    c2cMessage.setDeviceId("<device-id>");
    c2cMessage.setSourceId("source-id>");
    c2cMessage.setTargetId("<target-id>");
    c2cMessage.setMessageType("<message-type>");
    c2cMessage.setTtl(1);
    c2cMessage.setSystemId("<system-id>");
    c2cMessage.setSubSystemId("<sub-system-id>");
    c2cMessage.setCorrelationId("<correlation_id>");
    c2cMessage.setVersion("<version>");
    c2cMessage.setVin("<vin>");
    c2cMessage.setEcuType("<ecu-type>");
    c2cMessage.setTime(1625678536L);
    c2cMessage.setPropertyBag(propertyBag);
    c2cMessage.setBody("<body>");
    c2cMessage.setStatus("<status>");

    log.info("Message ready to publish: {}", c2cMessage);

    return c2cMessage;
  }
}
```

5.3 Stream Subscriber Sample Code

```
public class SampleStreamSubscriber {

  private static Logger log = LoggerFactory
          .getLogger(SampleStreamSubscriber.class);

  public static void main(String[] args) {

    // CREATE STREAM CONNECTION CONFIGURATION
    StreamClientConfig streamConnectionConfig = createStreamConnectionConfig();
    streamConnectionConfig
            .setStreamPublisherConfig(new StreamPublisherConfig());
    streamConnectionConfig
            .setStreamSubscriberConfig(new StreamSubscriberConfig());
    log.info("Stream Connection Config: {}", streamConnectionConfig);

    // CREATE STREAM CONFIGURATION
    StreamConfig<CommunicationCoreMessage> streamConfig = createStreamConfig();
    log.info("Stream Connection Config: {}", streamConfig);

    // CREATE STREAM CLIENT
    C2CStreamClient<CommunicationCoreMessage> streamClient = C2CStreamClientImpl
            .getC2CStreamClient(streamConnectionConfig);
    log.info("Stream Client created");

    // CREATE STREAM PROCESSOR
    C2CStreamMessageProcessor<CommunicationCoreMessage> streamProcessor = new C2CStreamMessageProcessor<>() {

      @Override
      public void processMessage(CommunicationCoreMessage message) {
        log.info("Message received: {}", message);
      }

      @Override
      public void processCorruptedMessage(String message) {
        log.info("Corrupted Message received: {}", message);
      }

    };

    // SUBSCRIBE TO STREAM
    streamClient.subscribe(streamProcessor, streamConfig);
    log.info("Subscribed to stream");

  }

  private static StreamClientConfig createStreamConnectionConfig() {
    String region = "<region>";
    return new StreamClientConfig(region);
  }

  private static StreamConfig<CommunicationCoreMessage> createStreamConfig() {

    StreamConfig<CommunicationCoreMessage> config = new StreamConfig<>(
            "<stream>", CommunicationCoreMessage.class, "<application_name>");
    log.info("Stream Config: {}", config);
    return config;
  }

}
```

6 Git Repositories

1. Stream Interface:
   Github-Enterpirse-India/c2c_base_stream_intf at develop
2. Stream Kinesis implementation:
   Github-Enterpirse-India/c2c_base_stream_kinesis_impl at develop