Qualcomm Car-to-Cloud Platform

Kinesis Wrapper Usage Document

Version No.3.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Jean Johnson		
Role	Developer		
Signature			
Date	June 11, 2021		

Table of Contents

1 Project Overview 3

2 Class Diagram 3

3 Interface Method Details 4

4 Usage Details 4

4.1 Dependencies 4

4.2 Stream Publisher Sample Code 5

4.3 Stream Subscriber Sample Code 7

5 Git Repositories 7

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

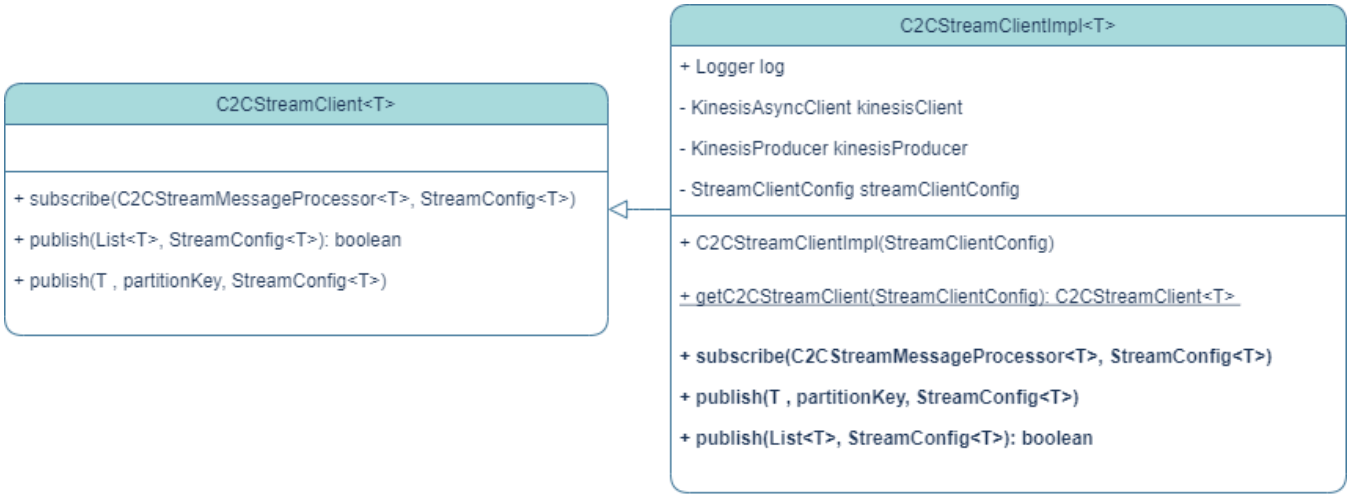
Retry Mechanism

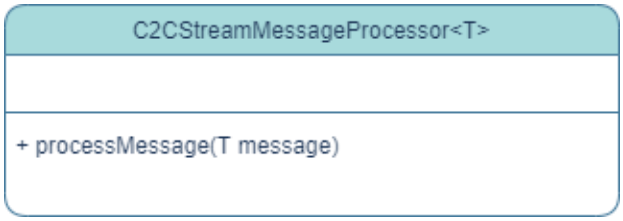
The Wrapper also executes Retry using Resilence4J Library. The *Number of attempt* and the *Interval* between the Attempt can be configured by the Client.

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 Interface Method Details

Method Name	Purpose	Input	Output
C2C Stream Client	Interface for sending and receiving messages from Stream		
subscribe()	Method to subscribe to Stream so that we can stream messages from the Stream.	1. C2CStreamMessageProcessor *(to process the messages)* 2. Stream Config *(configurations related to publish/subscribe)*	
publish()	Method to send a list of messages to Stream	1. List of messages to publish 2. Stream Config *(configurations related to publish/subscribe)*	
publish()	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	Method to process data received from Stream	Message received	

4 Usage Details

4.1 Dependencies

Stream Interface

```
<dependency>
	<groupId>com.c2c</groupId>
	<artifactId>c2c_base_stream_intf</artifactId>
	<version>0.0.4-SNAPSHOT</version>
</dependency>
```

Stream Kinesis implementation

```
<dependency>
	<groupId>com.c2c</groupId>
	<artifactId>c2c_base_stream_kinesis_impl</artifactId>
	<version>0.0.4-SNAPSHOT</version>
</dependency>
```

4.2 Stream Publisher Sample Code

```java
public class SampleStreamPublisher {

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

    public static void main(String[] args) {

        // CREATE STREAM CONNECTION CONFIGURATION
        StreamClientConfig streamConnectionConfig = createStreamConnectionConfig();
        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");

        // SET LIMIT FOR PUBLISH
        int limit = 3;

        List<CommunicationCoreMessage> messageList = new ArrayList<>();

        for (int i = 1; i <= limit; i++) {

            // CREATE MESSAGE
            CommunicationCoreMessage c2cMessage = createMessage(String.valueOf(Math.abs(new Random().nextInt(99999))));
            c2cMessage.setDeviceId("MULTIPLE-MESSAGE-TEST" + i);
            log.info("C2CCommunicationCoreMessage: {}", c2cMessage);

            messageList.add(c2cMessage);
        }
```

```
        log.info("Messages Published: {}", messageList);

        // PUBLISH LIST OF MESSAGE
        streamClient.publish(messageList, streamConfig);

        // CREATE MESSAGE
        CommunicationCoreMessage c2cMessage = createMessage(String.valueOf(Math.abs(new Random().nextInt(99999))));
        c2cMessage.setDeviceId("SINGLE-MESSAGE-TEST");
        log.info("C2CCommunicationCoreMessage: {}", c2cMessage);

        //PUBLISH SINGLE MESSAGE
        streamClient.publish(messageList.get(0), c2cMessage.getDeviceId(), streamConfig);

    }

    private static StreamClientConfig createStreamConnectionConfig() {
        String region = "us-east-1";
        return new StreamClientConfig(region);
    }

    private static StreamConfig<CommunicationCoreMessage> createStreamConfig() {

        StreamConfig<CommunicationCoreMessage> config = new StreamConfig<>("team1_demo_telemetry_stream1",
                CommunicationCoreMessage.class);
        log.info("Stream Config: {}", config);
        return config;
    }


    private static CommunicationCoreMessage createMessage(String random) {

        CommunicationCoreMessage c2cMessage = new CommunicationCoreMessage();

        Map<String, Object> propertyBag = new HashMap<>();
        propertyBag.put("systemName", "TCU");
        c2cMessage.setDeviceId("DEV-ID-" + random);
        c2cMessage.setMessageId("MSG-ID-" + random);
        c2cMessage.setUnitType("IVI");
        c2cMessage.setSourceId("SRC-ID-" + random);
        c2cMessage.setTargetId("DESTN-ID-" + random);
        c2cMessage.setMessageType("TELEMETRY EVENT");
        c2cMessage.setMessageCreationTime(new Timestamp(System.currentTimeMillis()));
        c2cMessage.setTtl(new Random().nextInt(100));
        c2cMessage.setPropertyBag(propertyBag);
        c2cMessage.setBody("CQkJInR5cGUiOiIiLA0KICAgICAgICAgICAgIm1");
        c2cMessage.setStatus("READY");

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

        return c2cMessage;
    }
}
```

4.3 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();
        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 PROCESSOR
        C2CStreamMessageProcessor<CommunicationCoreMessage> streamProcessor = (c2cMessage) -> {

            log.info("Message received: {}", c2cMessage);

        };

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

    }

    private static StreamClientConfig createStreamConnectionConfig() {
        String region = "us-east-1";
        return new StreamClientConfig(region);
    }

    private static StreamConfig<CommunicationCoreMessage> createStreamConfig() {

        StreamConfig<CommunicationCoreMessage> config = new StreamConfig<>("team1_demo_telemetry_stream1",
                CommunicationCoreMessage.class, "stream_client");
        log.info("Stream Config: {}", config);
        return config;
    }

}
```

5 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