DynamoDBStreams Kinesis Adapter V1->V2 Migration Notes

Andrew Muldowney - andrew.muldowney87@gmail.com

I tried my hand at converting the dynamodb-streams-kinesis-adapter from V1 to V2, AWS-SDK and Kinesis Client.

https://github.com/amuldowney/dynamodb-streams-kinesis-adapter

```
Java
 <properties>
        <aws-java-sdkv2.version>2.17.242</aws-java-sdkv2.version>
        <software-kinesis.version>2.5.4</software-kinesis.version>
        <aws.dynamodblocal.version>[1.12,2.0)</aws.dynamodblocal.version>
    </properties>
```

I was able to get all (existing) tests to pass which includes an integration test that writes some rows and reads them out of the stream. But during the process I had to either implement a `@KinesisClientInternalApi` or had to slow down the library in some way. I've noted the major ones here in this document. I hope this is helpful for others looking to do the same work and/or get some advice on ways to avoid implementing `@KinesisClientInternalApi` classes.

StreamsWorkerFactory.java

1. `ConfigsBuilder` doesn't take the `KinesisClientLibConfiguration` so we have to extract the specific configurations to override their settings.

```
Java
    StreamsWorkerFactory.java

    LeaseManagementConfig lmc = configsBuilder.leaseManagementConfig();
    lmc =
lmc.initialLeaseTableReadCapacity(config.getInitialLeaseTableReadCapacity());
```

```
    lmc =
lmc.initialLeaseTableWriteCapacity(config.getInitialLeaseTableWriteCapacity());
    lmc = lmc.shardSyncIntervalMillis(config.getShardSyncIntervalMillis());
```

2. `DynamoDBLeaseManagementFactory` is an `@KinesisClientInternalApi` but we need to override it to set our own DynamoDBStreamsShardSyncer because there is nowhere else we can do it. Creating a `DynamoDBLeaseManagementFactory` is a metric ton of constructor args. There is a place to set a hierarchicalShardSyncer on the `LeaseManagementConfig` but it doesn't work.

```java
StreamsWorkerFactory.java

LeaseManagementConfig lmc = configsBuilder.leaseManagementConfig();
lmc = lmc.hierarchicalShardSyncer(shardSyncer);//DOESNT WORK

//Instead we must do the following
 DynamoDBStreamsLeaseManagementFactory dynamoDBStreamsLeaseManagementFactory =
        new DynamoDBStreamsLeaseManagementFactory(
            streamsClient,
            dynamoDBClient,
             ...
            new DynamoDBStreamsShardSyncer(new StreamsLeaseCleanupValidator()),
             ...
            configsBuilder.retrievalConfig().streamTracker().isMultiStream() ?
                new DynamoDBMultiStreamLeaseSerializer() :
                new DynamoDBLeaseSerializer(),
             ...);
    lmc.leaseManagementFactory(dynamoDBStreamsLeaseManagementFactory);
HierarchicalShardSyncer
```

3. `HierarchicalShardSyncer` is an `@KinesisClientInternalApi` but we need to override it as well to support some specific DDB needs.

```java
class DynamoDBStreamsShardSyncer extends HierarchicalShardSyncer...

 * This class has been copied from ShardSyncer in KinesisClientLibrary and
edited slightly to enable DynamoDB Streams
```

```
 * specific behavior. It is a helper class to sync leases with shards of the
DynamoDB Stream.
 * It will create new leases/activities when it discovers new DynamoDB Streams
shards (bootstrap/resharding).
 * It deletes leases for shards that have been trimmed from DynamoDB Stream.
 * It also ensures that leases for shards that have been completely processed
are not deleted until their children
 * shards have also been completely processed.

software.amazon.awssdk.services.kinesis.model.Record
```

4. Thank goodness `SynchronousPrefetchingRetrievalFactory` exists, the default `RetrievalFactories` rely on stream connections to Kinesis which I didn't want to fake.

```java
 RetrievalConfig rc = configsBuilder.retrievalConfig();
    rc = rc.retrievalFactory(new SynchronousPrefetchingRetrievalFactory(

rc.streamTracker().streamConfigList().get(0).streamIdentifier().streamName(),
       //always a single stream tracker for dynamodb streams
        streamsClient,
        new DynamoDBStreamsRecordsFetcherFactory(),
        1000,
        executorService,
        300,
        Duration.ofSeconds(30)
    ));
```

Model package

1. All the model work is now static instantiation / mapping since V2 data classes are final w/ builder. This is maybe a little slower than before. I've renamed all the classes *Mapper to indicate they aren't adapting but creating new objects.

2. `AmazonServiceExceptionTransformer` work is extensive, mapping the new types along with going from editable to builders all over

3. RecordObjectMapper & RecordMapper have changed dramatically. Since we can't sub-class software.amazon.awssdk.services.kinesis.model.Record (its final now) we have to actually store the DDB record in the Kinesis record bytes and pull it out on the other end.

```java
     dynamoRecord = Record.builder()
        .awsRegion("us-east-1")
        .eventID(UUID.randomUUID().toString())
        .eventSource("aws:dynamodb")
        .eventVersion("1.1")
        .eventName(OperationType.MODIFY)
        .dynamodb(StreamRecord.builder()
            .approximateCreationDateTime(TEST_DATE)
            .keys(key)
            .oldImage(new HashMap<>(key))
            .newImage(newImage)
            .sizeBytes(Long.MAX_VALUE)
            .sequenceNumber(TEST_STRING)
            .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
            .build()).build();

        kinesisRecord = RecordMapper.convert(dynamoRecord);


    @Test
    public void testGetDataDeserialized() throws  IOException {
        Whitebox.setInternalState(RecordMapper.class, ObjectMapper.class,
MAPPER);

        java.nio.ByteBuffer dynamoRecordAsData =
kinesisRecord.data().asByteBuffer();
        Record actual =
MAPPER.readValue(BinaryUtils.copyBytesFrom(dynamoRecordAsData),
Record.serializableBuilderClass()).build();
        assertEquals(dynamoRecord, actual);
    }
```

ShardMapper

1. In ShardMapper we lie about the HashKeyRange to keep a different part of the code off our backs, parts of the KCL want to piece together hash keys which creates a loop that breaks the adapter.

Java
```java
public class ShardMapper {

    static final BigInteger MIN_HASH_KEY = BigInteger.ZERO;
    static final BigInteger MAX_HASH_KEY = new
BigInteger("2").pow(128).subtract(BigInteger.ONE);

    public static software.amazon.awssdk.services.kinesis.model.Shard
convert(software.amazon.awssdk.services.dynamodb.model.Shard internalShard) {
        return  software.amazon.awssdk.services.kinesis.model.Shard.builder()
            .shardId(internalShard.shardId())
            .parentShardId(internalShard.parentShardId())
            .adjacentParentShardId(null)

.hashKeyRange(software.amazon.awssdk.services.kinesis.model.HashKeyRange.builde
r()
                .startingHashKey(MIN_HASH_KEY.toString())
                .endingHashKey(MAX_HASH_KEY.toString())
                .build())
...
```