5

Kafka APIs for Oracle Transactional Event Queues

Oracle Transactional Event Queue (TxEventQ) makes it easy to implement event-based applications. It is also highly integrated with Apache Kafka, an open-source stream-processing software platform developed by LinkedIn and donated to the Apache Software Foundation, written in Scala and Java. Apart from enabling applications that use Kafka APIs to transparently operate on Oracle TxEventQ, Oracle TxEventQ also supports bi-directional information flow between TxEventQ and Kafka, so that changes are available in TxEventQ or Kafka as soon as possible in near-real-time.

Apache Kafka Connect is a framework included in Apache Kafka that integrates Kafka with other systems. Oracle TxEventQ will provide standard JMS package and related JDBC, Transaction packages to establish the connection and complete the transactional data flow. Oracle TxEventQ configures standard Kafka JMS connectors to establish interoperability and complete the data flow between the two messaging systems.

This chapter includes the following topics:

Apache Kafka Overview

Apache Kafka is a community distributed event streaming platform that is horizontally-scalable and fault-tolerant.

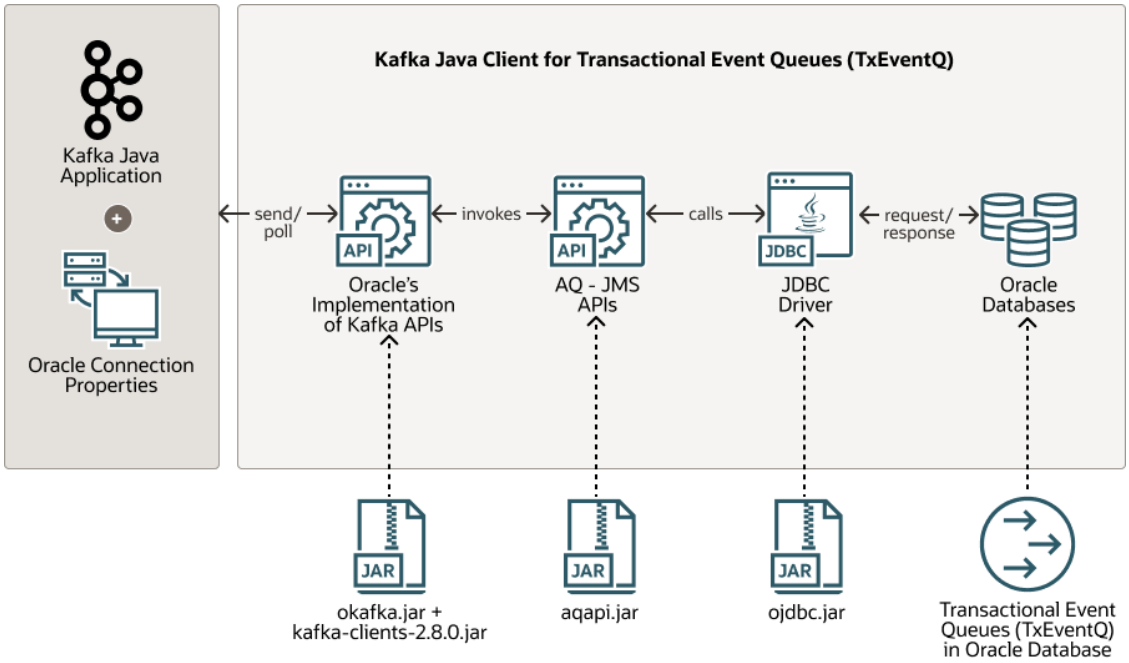
Kafka is deployed on a cluster of one or more servers. Each Kafka cluster stores streams of records in categories called topics. Each record consists of a key, a value, and a timestamp. The Kafka APIs allow an application to connect to a Kafka cluster and use the Kafka messaging platform.

Kafka Java Client for Transactional Event Queues

Oracle Database 21c introduced Kafka application compatibility with the Oracle Database. Oracle Database 23ai provides more refined compatibility for a Kafka application with the Oracle Database. This provides easy migration for Kafka Java applications to Transactional

Event Queues (TxEventQ). The Kafka Java APIs can now connect to an Oracle database server and use TxEventQ as a messaging platform.

Figure 5-1 Kafka Application Integration with Transactional Event Queue



This figure shows the Kafka API library, which contains Oracle specific implementation of Kafka's Java APIs which depends on the `kafka-clients-2.8.0.jar` file. This implementation internally invokes AQ-JMS APIs which in turn use the JDBC driver to communicate with the Oracle Database.

Developers can now migrate an existing Java application that uses Kafka to the Oracle database using `okafka.jar`. This client side library allows Kafka applications to connect to the Oracle Database instead of a Kafka cluster and use TxEventQ's messaging platform transparently.

Configuring Kafka Java Client for Transactional Event Queues

Prerequisites

The following are the prerequisites for configuring and running the Kafka Java client for TxEventQ in an Oracle Database.

1. Create a database user.
2. Grant the following privileges to the user.

> **Note:**
>
> It is preferred in general to assign or grant a specific quota on a tablespace to a database user instead of granting unlimited quota in default tablespace. One can create a table space and use the following command to grant quota on a specific tablespace to a database user.
>
> ```
> ALTER USER user QUOTA  UNLIMITED /* or size-clause */ on tablespace_name
> ```

- `GRANT EXECUTE on DBMS_AQ to user.`
- `GRANT EXECUTE on DBMS_AQADM to user.`
- `GRANT SELECT on GV_$SESSION to user;`
- `GRANT SELECT on V_$SESSION to user;`
- `GRANT SELECT on GV_$INSTANCE to user;`
- `GRANT SELECT on GV_$LISTENER_NETWORK to user;`
- `GRANT SELECT on GV_$PDBS to user;`
- `GRANT SELECT on USER_QUEUE_PARTITION_ASSIGNMENT_TABLE to user;`
- `exec DBMS_AQADM.GRANT_PRIV_FOR_RM_PLAN('user');`

3. Set the correct database configuration parameter to use TxEventQ.

```
SET STREAMS_POOL_SIZE=400M
```

> **Note:**
>
> Set the size appropriately based on your workload. `STREAMS_POOL_SIZE` cannot be set for Autonomous Database Shared. It is automatically configured. If set, then it is ignored.

4. Set `LOCAL_LISTENER` database parameter

```
SET LOCAL_LISTENER= (ADDRESS=(PROTOCOL=TCP)(HOST=<HOST_NAME.DOMAIN_NAME/ IP> )
(PORT=<PORT NUMBER>))
```

Connection Configuration

The Kafka API library connects to the Oracle Database using the JDBC Thin Driver. To set up this connection, the Kafka application can provide username and password in plain text, or applications can configure SSL. To run a Kafka application against Oracle Autonomous Database (ADB) on OCI, only SSL configuration is supported. For other deployments, you can connect to the Oracle Database using PLAINTEXT or SSL.

- PLAINTEXT: In this security protocol, JDBC connections to an Oracle Database are set up using the TCP protocol with username and password provided in plaintext in the `ojdbc.properties` file.

  To use the PLAINTEXT protocol, the application must set the following properties:

  - `oracle.service.name = <name of the service running on the instance>`
  - `bootstrap.servers = <host:port>`

- `security.protocol`=PLAINTEXT
- `oracle.net.tns_admin` = <*location of ojdbc.properties file*>

The following properties should be present in the `ojdbc.properties` file:

- `user` = <*nameofdatabaseuser*>
- `password` = <*userpassword*>

- SSL: To use the SSL security protocol to connect to an ATP database, perform the following additional steps.

  1. JDBC Thin Driver Connection prerequisites for SSL security:
     - JDK8u162 or higher.
     - `oraclepki.jar`, `osdt_cert.jar`, and `osdt_core.jar`
     - 18.3 or higher JDBC Thin Driver
  2. To leverage the JDBC SSL security to connect to the Oracle Database instance, the user has to provide the following properties. JDBC supports SSL secured connections to Oracle Database in two ways.
     - Using wallets. To use wallets:
       a. Add the required dependant jars for using Oracle Wallets in classpath.

          Download `oraclepki.jar`, `osdt_cert.jar`, and `osdt_core.jar` files along with the JDBC Thin Driver, and add these jars to classpath.

       b. Enable Oracle PKI provider

          Add `OraclePKIProvider` at the end of the file `java.security` (located at `$JRE_HOME/jre/lib/security/java.security`) if SSO wallet, that is, `cwallet.sso` is used for providing SSL security. For example:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=oracle.security.pki.OraclePKIProvider
```

          To use `ewallet.p12` for SSL security then place `OraclePKIProvider` before sun provider in file `java.security`. For example:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=oracle.security.pki.OraclePKIProvider
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=sun.security.jgss.SunProvider
security.provider.7=com.sun.security.sasl.Provider
```

       c. Set the following properties in the application.

```
security.protocol=SSL
oracle.net.tns_admin=<location of tnsnames.ora file>
```

```
tns.alias=<alias of connection string in tnsnames.ora>
```

Set the following properties in the `ojdbc.properties` file. This file must be available at the location specified by the `oracle.net.tns_admin` property.

```
user(in smallletters)=nameofdatabaseuser
password(in smallletters)=userpassword
oracle.net.ssl_server_dn_match=true
oracle.net.wallet_location="(SOURCE=(METHOD=FILE)
                                    (METHOD_DATA=(DIRECTORY=/
location../wallet_dbname)))"
```

– To use JDBC SSL security with a Java KeyStore, provide the following properties in the application:

```
security.protocol=SSL
oracle.net.tns_admin=<location of tnsnames.ora file>
tns.alias=<alias of connection string in tnsnames.ora>
```

Set the following properties in the `ojdbc.properties` file. This file must be available at the location specified by the `oracle.net.tns_admin` property.

```
user(in smallletters)=nameofdatabaseuser
password(in smallletters)=userpassword
oracle.net.ssl_server_dn_match=true
javax.net.ssl.trustStore==${TNS_ADMIN}/truststore.jks
javax.net.ssl.trustStorePassword=password
javax.net.ssl.keyStore=${TNS_ADMIN}/keystore.jks
javax.net.ssl.keyStorePassword=password
```

Note:

`tnsnames.ora` file in wallet downloaded from ATP contains a JDBC connection string which is used for establishing a JDBC connection.

See Also:

ALTER USER in *Oracle Database SQL Language Reference*

Kafka Client Interfaces

Kafka applications mainly use Producer, Consumer, and Admin APIs to communicate with a Kafka cluster. This version of Kafka client for TxEventQ supports only a subset of Apache Kafka 2.8.0's Producer, Consumer and Admin APIs and properties. With the `okafka.jar` client library, Kafka applications will be able to use the Oracle TxEventQ platform. The `okafka.jar` library requires JRE 9 or above.

We first illustrate the use of the Kafka Client APIs by way of simple examples, and later describe more details on the same.

Kafka API Examples

Example: Creating an Oracle Kafka Topic

```
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.admin.Admin;
import org.apache.kafka.clients.admin.CreateTopicsResult;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.common.KafkaFuture;

import org.oracle.okafka.clients.admin.AdminClient;

public class SimpleAdminOKafka {

    public static void main(String[] args) {
        Properties props = new Properties();
        //IP or Host name where Oracle Database 23ai is running and Database Listener's
Port
        props.put("bootstrap.servers", "localhost:1521");

        //name of the service running on the database instance
        props.put("oracle.service.name", "freepdb1");
        props.put("security.protocol","PLAINTEXT");

        // location for ojdbc.properties file where user and password properties are
saved
        props.put("oracle.net.tns_admin",".");

        try (Admin admin = AdminClient.create(props)) {
            //Create Topic named TEQ with 10 Partitions.
            CreateTopicsResult result = admin.createTopics(
                    Arrays.asList(new NewTopic("TEQ", 10, (short)0)));
            try {
                KafkaFuture<Void> ftr =  result.all();
                ftr.get();
            } catch ( InterruptedException | ExecutionException e ) {

                throw new IllegalStateException(e);
            }
            System.out.println("Closing  OKafka admin now");
        }
        catch(Exception e)
        {
            System.out.println("Exception while creating topic " + e);
            e.printStackTrace();
        }
    }
}
```

Example: Kafka Consumer API

```
import java.util.Properties;
import java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.common.header.Header;
import org.apache.kafka.common.TopicPartition;
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.oracle.okafka.clients.consumer.KafkaConsumer;

public class SimpleConsumerOKafka {

    // Dummy implementation of ConsumerRebalanceListener interface
    // It only maintains the list of assigned partitions in assignedPartitions list
    static class ConsumerRebalance implements ConsumerRebalanceListener {

        public List<TopicPartition> assignedPartitions = new ArrayList<>();

        @Override
        public synchronized void onPartitionsAssigned(Collection<TopicPartition>
partitions) {
            System.out.println("Newly Assigned Partitions:");
            for (TopicPartition tp :partitions ) {
                System.out.println(tp);
                assignedPartitions.add(tp);
            }
        }

        @Override
        public synchronized void onPartitionsRevoked(Collection<TopicPartition>
partitions) {
            System.out.println("Revoked previously assigned partitions. ");
            for (TopicPartition tp :assignedPartitions ) {
                System.out.println(tp);
            }
            assignedPartitions.clear();
        }
    }

    public static void main(String[] args) {
        //System.setProperty("org.slf4j.simpleLogger.defaultLogLevel", "TRACE");
        Properties props = new Properties();
        //IP or Host name where Oracle Database 23ai is running and Database Listener's
Port
        props.put("bootstrap.servers", "localhost:1521");

        //name of the service running on the database instance
        props.put("oracle.service.name", "freepdb1");
        props.put("security.protocol","PLAINTEXT");
```

```
        // location for ojdbc.properties file where user and password properties are
saved
        props.put("oracle.net.tns_admin",".");

        //Consumer Group Name
        props.put("group.id" , "CG1");
        props.put("enable.auto.commit","false");

        // Maximum number of records fetched in single poll call
        props.put("max.poll.records", 2000);

        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

        Consumer<String , String> consumer = new KafkaConsumer<String,
String>(props);
        ConsumerRebalanceListener rebalanceListener = new ConsumerRebalance();

        //Subscribe to a single topic named 'TEQ'.
        consumer.subscribe(Arrays.asList("TEQ"), rebalanceListener);

        int expectedMsgCnt = 40000;
        int msgCnt = 0;
        Instant startTime = Instant.now();
        try {
            while(true) {
                try {
                    //Consumes records from the assigned partitions of 'TEQ' topic
                    ConsumerRecords <String, String> records =
consumer.poll(Duration.ofMillis(10000));
                    //Print consumed records
                    for (ConsumerRecord<String, String> record : records)
                    {
                        System.out.printf("partition = %d, offset = %d, key = %s, value
=%s\n ", record.partition(), record.offset(), record.key(), record.value());
                        for(Header h: record.headers())
                        {
                            System.out.println("Header: " +h.toString());
                        }
                    }
                    //Commit all the consumed records
                    if(records != null && records.count() > 0) {
                        msgCnt += records.count();
                        System.out.println("Committing records " + records.count());
                        try {
                            consumer.commitSync();
                        }catch(Exception e)
                        {
                            System.out.println("Exception in commit " + e.getMessage());
                            continue;
                        }
                        if(msgCnt >= expectedMsgCnt )
                        {
                            System.out.println("Received " + msgCnt + " Expected " +
expectedMsgCnt +". Exiting Now.");
                            break;
                        }
                    }
                    else {
                        System.out.println("No Record Fetched. Retrying in 1 second");
```

```
                    Thread.sleep(1000);
                }

            }catch(Exception e)
            {
                System.out.println("Inner Exception " + e.getMessage());
                throw e;
            }
        }
    }catch(Exception e)
    {
        System.out.println("Exception from OKafka consumer " + e);
        e.printStackTrace();
    }finally {
        long runDuration = Duration.between(startTime, Instant.now()).toMillis();
        System.out.println("Closing OKafka Consumer. Received "+ msgCnt +" records.
Run Duration " + runDuration);
        consumer.close();
    }
  }
}
```

Example: Kafka Producer API

```
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.header.internals.RecordHeader;

import org.oracle.okafka.clients.producer.KafkaProducer;

import java.time.Duration;
import java.time.Instant;
import java.util.Properties;
import java.util.concurrent.Future;

public class SimpleProducerOKafka {
    public static void main(String[] args) {
        try {
            Properties props = new Properties();
            //IP or Host name where Oracle Database 23ai is running and Database
Listener's Port
            props.put("bootstrap.servers", "localhost:1521");

            //name of the service running on the database instance
            props.put("oracle.service.name", "freepdb1");
            props.put("security.protocol","PLAINTEXT");

            // location for ojdbc.properties file where user and password properties are
saved
            props.put("oracle.net.tns_admin",".");


            props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
            props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

            String baseMsg = "This is a test message ";
            // Creates OKafka Producer
```

```
            Producer<String, String> producer = new KafkaProducer<String, String>(props);

            Future<RecordMetadata> lastFuture = null;
            int msgCnt = 40000;
            Instant startTime = Instant.now();

            //Headers, common for all records
            RecordHeader rH1 = new RecordHeader("CLIENT_ID", "FIRST_CLIENT".getBytes());
            RecordHeader rH2 = new RecordHeader("REPLY_TO",
"REPLY_TOPIC_NAME".getBytes());

            //Produce 40000 messages into topic named "TEQ".
            for(int i=0;i<msgCnt;i++) {
                ProducerRecord<String, String> producerRecord = new
ProducerRecord<String, String>("TEQ", ""+i, baseMsg + i);
                producerRecord.headers().add(rH1).add(rH2);
                lastFuture =producer.send(producerRecord);
            }
            //Waits until the last message is acknowledged
            lastFuture.get();
            long runTime = Duration.between( startTime, Instant.now()).toMillis();
            System.out.println("Produced "+ msgCnt +" messages. Run Duration " +
runTime);
            //Closes the OKafka producer
            producer.close();
        }
        catch(Exception e)
        {
            System.out.println("Exception in Main " + e );
            e.printStackTrace();
        }
    }
}
```

Example: Deleting an Oracle Kafka Topic

```
import java.util.Collections;
import java.util.Properties;

import org.apache.kafka.clients.admin.Admin;

import org.oracle.okafka.clients.admin.AdminClient;

public class SimpleAdminDeleteTopic {

    public static void main(String[] args) {

        Properties props = new Properties();
        //IP or Host name where Oracle Database 23ai is running and Database Listener's
Port
        props.put("bootstrap.servers", "localhost:1521");

        //name of the service running on the database instance
        props.put("oracle.service.name", "freepdb1");
        props.put("security.protocol","PLAINTEXT");

        // location for ojdbc.properties file where user and password properties are
saved
        props.put("oracle.net.tns_admin",".");
```

```
        try (Admin admin = AdminClient.create(props)) {
            //Throws Exception if failed to delete the topic. Returns null on successful
deletion.
            org.apache.kafka.clients.admin.DeleteTopicsResult delResult =
                    admin.deleteTopics(Collections.singletonList("TEQ"));
            Thread.sleep(1000);
            System.out.println("Closing admin now");
        }
        catch(Exception e)
        {
            System.out.println("Exception while creating topic " + e);
            e.printStackTrace();
        }
    }

}
```

> **Note:**
>
> - Topics created using the KafkaAdmin interface can be accessed only by KafkaProducer or KafkaConsumer interfaces.
> - Similarly, the KafkaProducer and KafkaConsumer interfaces can produce or consume records only from topics which are created using the KafkaAdmin interface.

Kafka REST APIs for TxEventQ

The TxEventQ REST APIs allow common operations to produce and consume from topics and partitions, and are implemented using Oracle REST Data Services (ORDS) in the Oracle Database. Common operations include creating and deleting topics, producing and consuming messages, and operational APIs for getting consumer lag on a topic, seeking to an offset, among many others.

The following three APIs for Kafka allow TxEventQ to co-exist with Kafka deployments, and provide the advantages of transactional outbox, JMS messaging and pub/sub in the database and high throughput streaming of events to the event queue in the Oracle Database.

> **See Also:**
>
> Oracle Transactional Event Queues REST Endpoints for the Oracle REST Data Services API documentation

Overview of Kafka Producer Implementation for TxEventQ

Producer APIs allow a Kafka application to publish messages into Oracle Transactional Event Queues (TxEventQ). A Kafka application needs to provide Oracle specific properties: `bootstrap.servers`, `oracle.servicename`, and `oracle.net.tns_admin`. More details about these properties are mentioned in the configuration section. These properties are used to set

up the database connection and produce the message into TxEventQ. In the current release, Oracle's implementation of `KafkaProducer` supports only a subset of the Producer APIs.

Internally, an Oracle Kafka Producer object encapsulates an AQ JMS producer object which is used to publish messages into Oracle TxEventQ. Similar to Apache Kafka Producer, each Producer `send()` call will append a Kafka Record into a batch based on its topic and partition. Based on Apache Kafka's internal algorithm, a background thread will publish the entire batch to an Oracle TxEventQ.

The following `KafkaProducer` APIs are supported in Oracle Database 23ai.

- Constructor:

  `KafkaProducer`: Creates a producer object and internal AQ JMS objects. The `KafkaProducer` class has four types of constructors defined, which all take configuration parameters as input.

- Methods:
  - `send(ProducerRecord) , send(ProducerRecord, Callback)`:

    The `send` method asynchronously publishes a message into TxEventQ. This method returns immediately once a Kafka Record has been stored in the buffer of records waiting to be sent. If the buffer is full, then the send call blocks for a maximum time of `max.block.ms`. Records will be published into the topic using AQ JMS.

    The method returns a `Future<RecordMetadata>`, which contains the partition, offset, and publish timestamp of the record. Both the `send(ProducerRecord)` and `send(ProducerRecord, Callback)` versions are supported.

  - `close`: Closes the producer and frees the memory. It will also close the internal connection to the Oracle Database.

- Classes
  - `ProducerRecord`: A class that represents a message in the Kafka platform. The Kafka API library translates a `ProducerRecord` into a JMS BytesMessage for the TxEventQ platform.
  - `RecordMetadata`: This contains metadata of the record like topic, partition, offset, timestamp etc. of the Record in the Kafka platform. This is assigned values relevant for TxEventQs. A message id of TxEventQ is converted into an offset of `RecordMetadata`.
  - Callback Interface: A callback function which is executed once a Record is successfully published into a Kafka topic.
  - Partitioner Interface: Defines methods which map a `Key` of the message to a partition number of the topic. A partition number is analogous to a stream id of TxEventQs.

- Properties
  - `key.serializer` and `value.serializer`: Converts Key and payload into byte array respectively.
  - `acks`: For Kafka APIs, the only value relevant for the `acks` property is `all`. Any other field set by the user is ignored.
  - `linger.ms`: Time in miliseconds for which the sender thread waits before publishing the records in TxEventQ.
  - `batch.size`: Total size of records to be batched in bytes for which the sender thread waits before publishing records in TxEventQ.
  - `buffer.memory`: Total memory in bytes the accumulator can hold.

- `max.block.ms`: If `buffer.memory` size is full in the accumulator, then wait for `max.block.ms` amount of time before the `send()` method can receive an out of memory error.
- `retries`: This property enables a producer to resend a record in case of transient errors. This value limits the number of retries per batch.
- `retry.backoff.ms` : The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.
- `bootstrap.servers`: IP address and port of a machine where an instance of the database is running.

Overview of Kafka Consumer implementation for TxEventQ

The Consumer API allows applications to read streams of data from a Transactional Event Queue (TxEventQ). Kafka consumer for TxEventQ uses AQ JMS APIs and a JDBC driver to consume messages in batches from Oracle TxEventQ. For Oracle Kafka, consuming messages from a topic is achieved by dequeuing messages from a Transactional Event Queue.

Similar to Apache Kafka, in TxEventQ's implementation, a consumer group (subscriber) may contains many consumer instances (unique database sessions that are consuming for the subscriber). Each consumer group has a unique group-id (subscriber name). Each consumer instance internally maintains a single connection/session to an Oracle Database instance provided by the `bootstrap.servers` property. Oracle Database 23ai introduces Kafka API support for Consumer Group Rebalancing. Partitions of a topic will be distributed among the active consumers of a consumer group such that no two consumers from the same consumer group are assigned the same partition of the topic simultaneously. Whenever new consumers join a consumer group or an existing consumer leaves the group, the partitions will be redistributed among the active consumers.

For the 23ai release of Kafka APIs, a consumer can subscribe to only one topic.

The following `KafkaConsumer` APIs are supported in Oracle Database 23ai.

- Constructor: `KafkaConsumer`: Creates a consumer that allows the application to consume messages from a key based TxEventQ. Internal client side TxEventQ objects created are not visible to a client application. All variations of the `KafkaConsumer` constructor are supported in Oracle Database 23ai.
- Methods:
  - `Subscribe`: This method takes a list of topics to subscribe to. In Oracle Database 23ai, only the first topic of the list will be subscribed to. An exception is thrown if the size of the list is greater than 1. This method creates a durable subscriber on TxEventQ server side with Group-Id as subscriber name. An application can also implement the `ConsumerRebalanceListener` interface and pass an object of the implemented class to the subscribe method. This allows a consumer to execute callbacks when a partition is revoked or assigned.
  - `Poll`: The `poll` method returns a batch of messages from assigned partitions from TxEventQ. It attempts to dequeue a message from the key based TxEventQ for the subscriber. TxEventQ uses the array dequeue API of AQ JMS to receive a batch of messages from the queue. The size of the batch depends on the parameter `max.poll.records` set by the Kafka client application. `Poll` takes time in milliseconds as an argument. The AQ JMS API for array dequeue can pass this timeout as a

dequeue option to the TxEventQ server and make the dequeue call, which will wait for messages till the timeout if the full array batch is not complete.

When poll is invoked for the first time by a consumer, it triggers consumer rebalancing for all the alive consumers of the consumer group. At the end of the consumer rebalancing, all alive consumers are assigned topic partitions, and subsequent poll requests will fetch messages from the assigned partitions only.

An application can participate and influence rebalancing using the `ConsumerRebalanceListener` interface and `partition.assignment.strategy` configuration.

The `partition.assignment.strategy` configuration allows an application to select a strategy for assigning partitions to consumer streams. OKafka supports all values for this configuration parameter which are documented in the Apache Kafka 2.8.0 documentation.

The default value for this configuration is `org.oracle.okafka.clients.consumer.TXEQAssignor` which is aware of Oracle RAC and implements a strategy that is best for achieving higher throughput from Oracle TxEventQ.

This strategy prioritizes fair distribution of partitions and local consumption of messages while distributing partitions among alive sessions.

`ConsumerRebalanceListener` allows an application to invoke callbacks when partitions are revoked or assigned to a consumer.

The database view `USER_QUEUE_PARTITION_ASSIGNMENT_TABLE` allows a developer to view the current distribution of partitions among the alive consumers.

- `commitSync`: Commits all consumed messages. Commit to an offset is not supported in Oracle Database 23ai. This call directly calls commit on the database which commits all consumed messages from TxEventQ.
- `commitAsync`: This call is translated into `commitSync`. A callback function passed as an argument gets executed once the commit is successful.
- `Unsubscribe`: Unsubscribes the topic that a consumer has subscribed to. A consumer can no longer consume messages from unsubscribed topics. This call does not remove a subscriber group from the TxEventQ metadata. Other consumer applications can still continue to consume for the same consumer group.
- `close`: Closes the consumer and unsubscribes the topic it has subscribed to.

- Class: `ConsumerRecord`: A class that represents a consumed record in the Kafka platform. Kafka APIs receive AQ JMS messages from TxEventQ and convert each of them into a `ConsumerRecord` and deliver it to the application.
- Properties:
  - `auto.offset.reset`: When there is no initial offset found for this consumer group, then the value of this property controls whether to consume the messages from the beginning of the topic partition or to only consume new messages. Values for this property and its usage are as follows:
    - `earliest`: Consume from the beginning of the topic partition.
    - `latest`: Consume from the end of the topic partition (default).
    - `none`: Throw an exception if no offset present for the consumer group.
  - `key.deserializer` and `value.deserializer`: For Oracle Kafka messaging platform, key and value are stored as byte arrays in a JMS Message in Oracle's TxEventQ. On

consuming, these byte arrays are deserialized into a key and a value using `key.deserializer` and `value.deserializer` respectively. These properties allow an application to convert `Key` and `Value`, which are stored in byte array format, into application specified data types.

  - `group.id`: This is a consumer group name for which messages are consumed from the Kafka topic. This property is used as a durable subscriber name for key based TxEventQs.
  - `max.poll.records`: Maximum number of records to fetch in a single array dequeue call from an Oracle TxEventQ server.
  - `enable.auto.commit`: Enables auto commit of consumed messages for every specified interval.
  - `auto.commit.interval.ms`: Interval in milliseconds for auto commit of messages.
  - `bootstrap.servers`: IP address and port of a machine where a database instance is running.

Overview of Kafka Admin Implementation for TxEventQ

The Kafka admininistrative API allows applications to perform administrative tasks like creating a topic, deleting a topic, adding a partition to a topic and so on. Oracle Database 23ai release supports only the following administrative APIs.

- Methods
  - `create(props)` and `create(config)`: Creates an object of `KafkaAdmin` class that uses passed parameters. The method creates a database session which is used for further operations. An application has to provide connection configuration parameters as explained in the Connection Configuration section.
  - `createTopics()`: Allows an application to create a Kafka Topic. This creates a TxEventQ in the user's schema.
  - `close()`: Closes a database session and Admin client.
  - `deleteTopic`: Deletes a Kafka Topic. This returns `null` when a topic is deleted successfully. Otherwise, the method throws an exception. The method does not return until the topic is successfully deleted or any error is encountered.
- Classes: `NewTopic`: Class used for creating a new topic. This class contains parameters with which a transactional event queue is created.

Kafka REST APIs for TxEventQ

The TxEventQ REST APIs allow common operations to produce and consume from topics and partitions, and are implemented using Oracle REST Data Services (ORDS) in the Oracle Database. Common operations include creating and deleting topics, producing and consuming messages, and operational APIs for getting consumer lag on a topic, seeking to an offset, among many others.

The following three APIs for Kafka allow TxEventQ to co-exist with Kafka deployments, and provide the advantages of transactional outbox, JMS messaging and pub/sub in the database and high throughput streaming of events to the event queue in the Oracle Database.

> **See Also:**
>
> Oracle Transactional Event Queues REST Endpoints for the Oracle REST Data Services API documentation

Kafka Connectors for TxEventQ

The Kafka Sink and Source Connector requires a minimum Oracle Database version of 21c in order to create a Transactional Event Queue. To use the application, Kafka with a minimum version number of 3.1.0 will need to be downloaded and installed on a server.

> **See Also:**
>
> https://github.com/oracle/okafka/tree/master/connectors for more information.

Monitoring Message Transfer

The Sink/Source connector messages transfer can be monitored from Oracle TxEventQ.

> **See Also:**
>
> Monitoring Transactional Event Queues to startup TxEventQ Monitor System to check enqueue/dequeue rate, TxEventQ depth, and more DB/System Level statistics.