

# Oracle® Banking Liquidity Management Kafka Configuration Guide



Release 14.7.4.0.0  
G10125-01  
June 2024

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Copyright © 2018, 2024, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

# Contents

## Preface

---

Purpose	v
Audience	v
Documentation Accessibility	v
Diversity and Inclusion	v
Related Resources	vi
Acronyms and Abbreviations	vi

## 1 Prerequisites

---

## 2 Kafka Middleware Setup

---

2.1 Zookeeper Setup	2-1
2.2 Kafka Setup	2-2

## 3 Important Commands

---

## 4 Increase Replication Factor for an existing topic

---

## 5 Security - SSL Encryption with SASL-SCRAM Authentication

---

## 6 Implementation

---

## 7 Flow Diagram

---

## 8 Payload and Header

---

## 9 Tables

---

### Index

---

# Preface

- [Purpose](#)
- [Audience](#)
- [Documentation Accessibility](#)
- [Diversity and Inclusion](#)
- [Related Resources](#)
- [Acronyms and Abbreviations](#)

## Purpose

This guide provides the information about the kafka implementation which allows the user to publish and consume message from/by publisher and consumer respectively.

## Audience

This guide is intended for the implementation teams.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### **Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

---

## Related Resources

For more information on any related features, refer to the following documents:

- *Oracle Banking Liquidity Management Installation Guide*
- *Oracle Banking Liquidity Management Configuration Guide*

## Acronyms and Abbreviations

The list of the acronyms and abbreviations used in this guide are as follows:

**Table 1 Acronyms and Abbreviations**

Abbreviation	Description
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface

# 1

## Prerequisites

This topic provides the prerequisites to be performed before the kafka configuration.

The following installation should be completed and running to enable the APIs to publish and consume message from Kafka.

- Zookeeper
- Kafka

Minimum requirements for installation are:

- Partition count: 2
- Replication factor: 2
- Kafka brokers: 2
- Zookeeper nodes: 2
- Servers: 2

These values can be increased based on the requirement and load. Restrict the access to the server\*.properties file of Kafka servers.

# 2

## Kafka Middleware Setup

This topic provides the information about the kafka middleware setup.

- [Zookeeper Setup](#)  
This topic provides the systematic instructions to install and setup the Zookeeper.
- [Kafka Setup](#)  
This topic provides the systematic instruction to install and setup kafka.

### 2.1 Zookeeper Setup

This topic provides the systematic instructions to install and setup the Zookeeper.

Kafka uses ZooKeeper to manage the cluster. ZooKeeper is used to coordinate the brokers/ cluster topology. ZooKeeper is a consistent file system for configuration information. ZooKeeper gets used for leadership election for Broker Topic Partition Leaders. Here we are going to start a node of 2 zookeeper ensemble on 2 servers each.

1. Extract the kafka installation files in `/tools/kafka` on both the servers.
2. Navigate to config folder in `/tools/kafka/conf`.
3. Duplicate the `zoo_sample.cfg` and rename it to `zookeeper1.cfg`
4. Open `zookeeper1.cfg` and modify the following properties.

```
DataDir= <kafka home directory>/data
tickTime=2000
clientPort= Zookeeper client Port value (2181)
initLimit=10
syncLimit=5

server.1=<hostname>:<peer port>:<leader port>
#1 is the id that we put in myid file.

server.2=<hostname>:<peer port>:<leader port>
#2 is the id that we will put in myid file of second node.

server.3=<hostname>:<peer port>:<leader port>
#3 is the id that we will put in myid file of third.
```

#### Example:

```
tickTime=2000

initLimit=5

syncLimit=2

clientPort=2181

dataDir=/tmp/zookeeper-oblm/zookeeper-node1
```



```
server.1=server1-IP:2666:3666
server.2=server2-IP:2667:3667
```

 **Note:**

Update the IP value with the respective server IP.

- Duplicate the **zoo.cfg** file and rename it as **zookeeper2.cfg** in the same directory on Server 2 (Other names can also be used). These configuration files used for each of the zookeeper nodes
- Open **zookeeper2.cfg** and modify the following properties.

```
clientPort=2182
dataDir=/tmp/zookeeper-oblm/zookeeper-node2
server.1=server1-IP:2666:3666
server.2=server2-IP:2667:3667
```

 **Note:**

Update the IP value with the respective server IP.

- Copy the **zookeeper1.cfg** and **zookeeper2.cfg** and Paste it in the local.
- Open the directory `/tmp/zookeeper-oblm/zookeeper-node1` on server 1 and create a file named **myid**, open with text editor and write 1, save and close.
- Open the directory `/tmp/zookeeper-oblm/zookeeper-node2` on server 2 and create a file named **myid**, open with text editor and write 2, save and close.
- Run the command to start the zookeeper nodes.

On Server 1:

```
nohup ./bin/zkServer.sh start conf/zookeep
```

On Server 2:

```
nohup ./bin/zkServer.sh start conf/zookeep
```

## 2.2 Kafka Setup

This topic provides the systematic instruction to install and setup kafka.

- Extract the kafka installation file in `/tools/kafka` on both the servers.
- Navigate to config folder in Apache Kafka (`/tools/kafka/config`).
- Duplicate the **server.properties** from config folder and rename it to **server1.properties**.
- Open **server1.properties** and modify the following properties.

```
broker.id= (Unique Integer which identifies the kafka broker in the
cluster.
listeners=PLAINTEXT://<hostname>:<Kafka broker listen port(9092)>
```

```
log.dirs=<Kafka home directory>/logs
log.retention.hours= <The number of hours to keep a log file
before deleting it (in hours), tertiary to log.retention.ms property>
log.retention.bytes= <The maximum size of the log before deleting it>
log.segement.bytes= <The maximum size of a single log file>
log.retention.check.interval.ms= <The frequency in milliseconds that
the log cleaner checks whether any log is eligible for deletion>
zookeeper.connect=<zookeeper_hostname_1>:<zookeeper_client_port>,
<zookeeper_hostname_2>:<zookeeper_client_port>,<zookeeper_hostname_3>:<zook
eeper_client_port>
```

**Example:**

```
broker.id=0

port=9092

log.dirs=/tmp/kafka-oblm/logs-node1

zookeeper.connect=server1-IP:2181,server2-IP:2182

num.partitions=2

min.insync.replicas=1

default.replication.factor=2

offsets.topic.replication.factor=2

transaction.state.log.replication.factor=2

transaction.state.log.min.isr=1
```

 **Note:**

If the Apache Zookeeper is on different server, then change the `zookeeper.connect` property. i.e., update the highlighted value for the respective server IPs. ***min.insync.replicas***: A typical configuration is replication-factor minus 1.

5. Duplicate the **server.properties** into the same directory and rename it to **server2.properties** on server 2.
6. Open **server2.properties** and modify the following properties.

```
broker.id=1

broker.id=1

log.dirs=/tmp/kafka-oblm/logs-node2
```

 **Note:**

By default, Apache Kafka will run on port 9092 and Apache Zookeeper will run on port 2181.

7. Copy the **server1.properties** and **server2.properties** and paste it in local.

8. To run Kafka brokers, change path to `/tools/kafka` directory and run the following command in separate terminals.

On Server 1:

```
nohup ./bin/kafka-server-start.sh config/server1.properties
```

On Server 2:

```
nohup ./bin/kafka-server-start.sh config/server2.properties
```

9. The values set for Logs is under the segment: "Log Retention Policy" in `server*.properties` file attached in the document. The values set under this segment are defaults from Apache
10. At present, kafka takes the default value for message size as:  
`message.max.bytes=1000012`
11. Add and update this field in `server*.properties` for increasing based on requirement.
12. To add compression type for all data generated by the producer, add the following property in `server*.properties` file.

```
compression.type=none
```

 **Note:**

The default is none (i.e. no compression). Valid values are none, gzip, snappy, lz4, or zstd.

# 3

## Important Commands

This topic provides the information about the important commands used for Kafka configuration.

### View the Topic Configurations

```
./kafka-topics.sh --describe --zookeeper zookeeper-server --topic topic-name
```

#### Example:

```
./kafka-topics.sh --describe --zookeeper localhost:2181 --topic structure-closed
```

#### Output:

```
Topic: structure-closed PartitionCount: 2 ReplicationFactor: 2 Configs:  
Topic: structure-closed Partition: 0 Leader: 1 Replicas: 1,0 Isr: 1,0  
Topic: structure-closed Partition: 1 Leader: 0 Replicas: 0,1 Isr: 0,1
```

### View the Messages Sent from producer-consumer

```
./kafka-console-consumer.sh --bootstrap-server Kafka-server --topic topic-name
```

#### Example:

```
./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic structure-closed
```

### Create Kafka Topics manually

```
./kafka-topics.sh --create --bootstrap-server kafka-server --replication-factor  
factor-value --partitions partition-value --topic topic-name
```

#### Example:

```
./kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor  
2 --partitions 2 --topic structure-closed
```

If the topics are created manually before the microservice deployment, then the values in the above command is considered otherwise if we are depending on the microservice deployment then the values configured in the server.properties file of Kafka is considered when the topics are created.

Configurations pertinent to topics have both a server default as well an optional per-topic override. If no per-topic configuration is given the server default is used. The override can be set at topic creation time by giving one or more --config options.

# 4

## Increase Replication Factor for an existing topic

This topic provides the systematic instruction to increase Replication Factor for an existing topic.

In case, a topic is already created, and the user want to increase the replication factor. Then, follow the below steps. Explanation is given below with an example and desired output for easier understanding.

Increase the replicas for the topic structure-closed in partition 0 from only on broker id 0 to broker id 0, 1. i.e. increase replication factor of 1 to 2.

1. Download a [Increasing Replication Factor](#) file and save to the local.
2. Command to increase the replication factor.

```
./kafka-reassign-partitions.sh --zookeeper zookeeper-server --reassignment-json-file jsonFilePath --execute
```

**Example:**

```
./kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file D:\kafka\kafka_2.12-2.3.1\config\increase-replication-factor.json --execute
```

**Output:**

Current partition replica assignment

```
{"version":1,"partitions":[{"topic":"structure-closed","partition":1,"replicas":[0],"log_dirs":["any"]},{"topic":"structure-closed","partition":0,"replicas":[1],"log_dirs":["any"]}]}
```

Save this to use as the --reassignment-json-file option during rollbackSuccessfully started reassignment of partitions.

3. Command to increase the replication factor.

```
./kafka-reassign-partitions.sh --zookeeper zookeeper-server --reassignment-json-file jsonFilePath --execute
```

**Example:**

```
./kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file D:\kafka\kafka_2.12-2.3.1\config\increase-replication-factor.json --execute
```

**Output:**

Current partition replica assignment:

```
{"version":1,"partitions":[{"topic":"structure-closed","partition":1,"replicas":[0],"log_dirs":["any"]},{"topic":"structure-closed","partition":0,"replicas":[1],"log_dirs":["any"]}]}
```

Save this to use as the --reassignment-json-file option during rollbackSuccessfully started reassignment of partitions.

---

4. Command to check the status of the partition reassignment.

```
./kafka-reassign-partitions.sh --zookeeper zookeeper-server --reassignment-  
json-file jsonFilePath --verify
```

**Example:**

```
./kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-  
file D:\kafka\kafka_2.12-2.3.1\config\increase-replication-factor.json -verify
```

**Output:**

Status of partition reassignment:

Reassignment of partition structure-closed-0 completed successfully.

5. Describe and check the topic.

```
./kafka-topics.sh --describe --zookeeper zookeeper-server --topic topic-name
```

**Example:**

```
./kafka-topics.sh --describe --zookeeper localhost:2181 --topic structure-  
closed
```

**Output:**

Topic: structure-closed PartitionCount: 2 ReplicationFactor: 1 Configs:

Topic: structure-closed Partition: 0 Leader: 1 Replicas: 0,1 Isr: 1,0

Topic: structure-closed Partition: 1 Leader: 0 Replicas: 0 Isr: 0

# 5

## Security - SSL Encryption with SASL-SCRAM Authentication

This topic describes about Security - SSL Encryption with SASL-SCRAM authentication.

### Generate Keystore

The items highlighted in bold are placeholders and should be replaced with suitable values when running the command.

```
keytool -genkeypair -alias alias -keyalg keyalg -keysize keysize -sigalg sigalg -validity valDays -keystore keystore
```

**Table 5-1 Generate Keystore - Keyword Details**

Keyword	Description
alias	Used to identify the public and private key pair created.
keyalg	It is a key algorithm used to generate the public and private key pair. The RSA key algorithm is recommended.
keysize	It is the size of the public and private key pairs generated. A key size of 1024 or more is recommended. Please consult with your CA on the key size support for different types of certificates.
sigalg	It is the algorithm used to generate the signature. This algorithm should be compatible with the key algorithm and should be one of the values specified in the Java Cryptography API Specification and Reference.
valdays	It is the number of days for which the certificate is to be considered valid. Please consult with your CA on this period.
keystore	It is used to specify the location of the JKS file. If no JKS file is present in the path provided, one will be created.

The command prompts for the following attributes of the certificate and Keystore:

**Table 5-2 Generate Keystore - Attributes**

Attributes	Description
<b>Keystore Password</b>	Specify a password used to access the Keystore. This password needs to be specified later when configuring the identity store in Kafka server.
<b>Key Password</b>	Specify a password used to access the private key stored in the Keystore. This password needs to be specified later when configuring the SSL attributes of the Kafka Server.

**Table 5-2 (Cont.) Generate Keystore - Attributes**

Attributes	Description
<b>First and Last Name (CN)</b>	Enter the domain name of the machine used to access Oracle Banking Liquidity Management. For example, <a href="http://www.example.com">www.example.com</a> .
<b>Name of your Organizational Unit</b>	The name of the department or unit making the request. Use this field to further identify the SSL Certificate you are creating, for example, by department or by physical server.
<b>Name of your Organization</b>	The name of the organization making the certificate request. For example, Oracle Financial Services. It is recommended to use the company or organization's formal name, and this name entered here must match the name found in official records.
<b>Name of your City or Locality</b>	The city in which your organization is physically located. For example, Bengaluru.
<b>Name of your State or Province</b>	The state/province in which your organization is physically located. For example, Karnataka.
<b>Two-letter Country Code for this Unit</b>	The country in which your organization is physically located. For example, US, UK, IN, etc.

**Example 5-1 Sample Execution**

Listed below is the result of a sample execution.

```
keytool -genkeypair -alias OBLMcert -keyalg RSA -keysize 1024 -sigalg
SHA512withRSA
-validity 365 -keystore D:\kafka\securityKeys\KafkaServerKeystore.jks
```

Enter keystore password:<Enter a password to protect the keystore>

Re-enter new password:<Confirm the password keyed above>

What is your first and last name?

[Unknown]: name.oracle.com

What is the name of your organizational unit?

[Unknown]: OBLM

What is the name of your organization?

[Unknown]: Oracle Financial Services

What is the name of your City or Locality?

[Unknown]: Bengaluru

What is the name of your State or Province?

[Unknown]: Karnataka

What is the two-letter country code for this unit?

[Unknown]: IN

Is CN= name.oracle.com, OU=OBLM, O=Oracle Financial Services, L= Bengaluru, ST= Karnataka, C=IN correct? [no]: yes



Enter key password for < OBLMcert >

RETURN if same as keystore password): <Enter a password to protect the key>

Re-enter new password: <Confirm the password keyed above>

### Export Private Key as Certificate

Export private key as certificate command is mentioned below:

```
keytool -export -alias <alias_name> -file
<export_certificate_file_name_with_location.cer>
-keystore <keystore_name.jks> -keypass <Private key Password> -storepass
<Store Password>
```

#### Example:

```
keytool -export -alias OBLMcert -file D:\kafka\securityKeys\KafkaCert.cer
-keystore D:\kafka\securityKeys\KafkaServerKeystore.jks -keypass oracle123 -
storepass oracle123
```

If successful, the following message will be displayed:

Certificate stored in file < KafkaCert.cer>

### Import the Certificate and Generate Trust Store

To import the certificate and generate Trust store, the command is mentioned below:

```
keytool -import -alias alias -file cert_file -keystore truststore -storepass
storepass
```

**Table 5-3 Generate Trust Store - Keyword Details**

Keyword	Description
alias	It is used to identify the public and private key pair. Specify the alias of the key pair used to create the CSR in the earlier step.
cert_file	It is the location of the file containing the PKCS#7 formatted reply from the CA, containing the signed certificate.
truststore	It is the location where the TrustStore should be generated.
storepass	It is the password for the TrustStore.

The user can generate two TrustStores from the same cert.

- One used for Kafka server
- One used for Clients

**Example:**

```
keytool -import -alias OBLMcert -file D:\kafka\securityKeys\KafkaCert.cer
-keystore D:\kafka\securityKeys\KafkaServerTrustStore.jks -storepass oracle123
```

```
keytool -import -alias OBLMcert -file D:\kafka\securityKeys\KafkaCert.cer
-keystore D:\kafka\securityKeys\KafkaClientTrustStore.jks -storepass oracle123
```

Three Keystore files are required for this method as given in the table below:

**Table 5-4 Keystore Files**

File Name	Description
KafkaServerKeystore.jks	Keystore file for Kafka brokers
KafkaServerTrustStore.jks	TrustStore file for server
KafkaClientTrustStore.jks	TrustStore file for client

To validate the server, each client should import the `KafkaClientTrustStore.jks` file.

 **Note:**

The truststore files should be generated using the same CA. The user can generate and place these files on all the different servers of Kafka so that they can be accessed by `server*.properties` file. The `KafkaClientTrustStore.jks` should be placed on the server, which is accessible by the microservices also.

**Create Users in Zookeeper**

To create users in Zookeeper, follow below steps:

1. Start the zookeeper.

 **Note:**

Refer to [Zookeeper Setup](#) topic.

2. Follow the below steps for user creation.
  - a. Execute the admin command for admin user creation.

```
./kafka-configs.sh --zookeeper localhost:2181,localhost:2182 --alter --
add-config
"SCRAM-SHA-256=[password=admin-secret],SCRAM-SHA-512=[password=admin-
secret]"
--entity-type users --entity-name admin
```

 **Note:**

The user created with admin as username and password is setup for the user for each scram mechanism. Here, the user **admin** is used for Kafka broker auth.

- b. Execute the test command for test user creation.

```
./kafka-configs.sh --zookeeper localhost:2181,localhost:2182 --alter --
add-config
"SCRAM-SHA-256=[iterations=8192,password=alice-secret],SCRAM-
SHA-512=[password=alice-secret]"
--entity-type users --entity-name alice
```

 **Note:**

The user created with alice as username and password is setup for the user for each scram mechanism. Here, the user **alice** is used for client auth. For multiple zookeeper nodes, use comma separated serverIP:port like in the above example(localhost:2181,localhost:2182).

## Configure Brokers

Some modifications need to be made in the server\*.properties file of kafka server. The following properties need to be added in **server1.properties** file of kafka.

```
##### SSL-SCRAM Settings #####
ssl.endpoint.identification.algorithm=
ssl.truststore.location=D:\\kafka\\securityKeys\\KafkaServerTrustStore.jks
ssl.truststore.password=oracle123
ssl.keystore.location=D:\\kafka\\securityKeys\\KafkaServerKeystore.jks
ssl.keystore.password=oracle123
ssl.key.password=oracle123
sasl.enabled.mechanisms= SCRAM-SHA-256
sasl.mechanism.inter.broker.protocol= SCRAM-SHA-256
security.inter.broker.protocol=SASL_SSL
listeners=SASL_SSL://HOSTNAME:9092
advertised.listeners=SASL_SSL://IP:9091
listener.name.sasl_ssl.scram-sha-256
.sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required
username="admin" password="admin-secret";
```

 **Note:**

In the highlighted section, give the absolute path of the Kafka Server Truststore and keystore, and its respective passwords. Modify the hostname and IP in the listeners and advertised.listeners properties field accordingly

Copy the above properties into the **server2.properties** file and modify the hostname/IP and port in the listeners and advertised.listeners properties field. Sample properties files can be downloaded through the below link.

Download server1.properties and server1.properties and save to the local.

Start the kafka servers.



### Note:

Refer to [Kafka Setup](#) topic.

## Changes to Clients

For the microservices which publish/consume data through kafka, insert the following values in the PROPERTIES table in PLATO schema before deployment.

**Table 5-5 PLATO PROPERTIES Table - Key Values**

KEY	VALUE
plato.services.kafka.brokers	<comma separated kafka hostname:port>
plato.services.zknodes	<comma separated Zookeeper hostname:port>
plato.services.kafka.security.protocol	SASL_SSL
plato.services.kafka.truststore.location	<absolute path of client truststore>
plato.services.kafka.truststore.password	<encrypted truststore password>
spring.cloud.stream.kafka.binder.configuration.sasl.mechanism	SCRAM-SHA-256
spring.cloud.stream.kafka.binder.jaas.loginModule	org.apache.kafka.common.security.scram.ScramLoginModule
spring.cloud.stream.kafka.binder.jaas.options.userName	<Zookeeper SCRAM user created for clients>
spring.cloud.stream.kafka.binder.jaas.options.password	<Zookeeper SCRAM user encrypted password for clients>

To encrypt the password, use the following api of plato-config-service of Oracle Banking Liquidity Management:

API: `http://hostname:port/config-service/encrypt`

Request Type: Text

Request Body: Password

### Example 1:

Once the above API is hit for the following passwords, the response of encrypted value is received.

alice-secret : 2f32dc1770acec085105e3ba585cc44c71534451b88b6047504f11191ad8cc1f  
 oracle123 : 7ec1250634259a1af12f74a7e4705ade7493a4695cc1efd3b713571453fda266

### Example 2:

When inserting to properties table, append the encrypted values with the keyword {cipher} to get it decrypted by the config-service during fetch as given in example below.

```
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10110,'oblm-structure-
```

```

services','jdbc','jdbc','plato.services.kafka.brokers','localhost:9092,localhost:
9093');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10111,'oblm-structure-
services','jdbc','jdbc','plato.services.zknodes','localhost:2181');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10112,'oblm-structure-
services','jdbc','jdbc','plato.services.kafka.security.protocol','SASL_SSL');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10113,'oblm-structure-
services','jdbc','jdbc','plato.services.kafka.truststore.location','D:\kafka\secu
rityKeys\KafkaClientTrustStore.jks');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10114,'oblm-structure-
services','jdbc','jdbc','plato.services.kafka.truststore.password','{cipher}7ec12
50634259a1af12f74a7e4705ade7493a4695cc1efd3b713571453fda266');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10115,'oblm-structure-
services','jdbc','jdbc','spring.cloud.stream.kafka.binder.configuration.sasl.mech
anism','SCRAM-SHA-256');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10116,'oblm-structure-
services','jdbc','jdbc','spring.cloud.stream.kafka.binder.jaas.loginModule','org.
apache.kafka.common.security.scram.ScramLoginModule');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10117,'oblm-structure-
services','jdbc','jdbc','spring.cloud.stream.kafka.binder.jaas.options.username',
'alice');
insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10118,'oblm-structure-
services','jdbc','jdbc','spring.cloud.stream.kafka.binder.jaas.options.password',
'{cipher}2f32dc1770acec085105e3ba585cc44c71534451b88b6047504f11191ad8cc1f');

```

### Important Commands

Create Topics manually is same as the command mentioned in [Create Kafka Topics Manually](#). If the user want to view the messages getting sent in kafka, then store the below lines in a file and name it as **ssl.properties**.

```

ssl.truststore.location=D:\\kafka\\securityKeys\\KafkaClientTrustStore.jks
ssl.truststore.password=oracle123
security.protocol=SASL_SSL
ssl.endpoint.identification.algorithm=
sasl.mechanism=SCRAM-SHA-256
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required
\username="alice"
\password="alice-secret";

```

#### Note:

Update the trust store location and password.

Download ssl.properties file and save to the local.

Command to view the messages being published:

```
./kafka-console-consumer.sh --bootstrap-server kafka-server --topic topicName --  
consumer.config absolute-path-of-consumer-config --from-beginning
```

**Example:**

```
./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic oblm --  
consumer.config D:\kafka\kafka_2.12-2.3.1\config\ssl.properties --from-beginning
```

# 6

## Implementation

This topic describes the implementation flow for the various service functionalities.

### The Flow

There is an events table in Maintenance schema with all the events that we will publish listed on it with some more properties. There is an IsEnabled column for all the events listed. Only for all those events where the IsEnabled field is set to true will publish to kafka.

oblm-services that wants to publish to kafka will fetch the events table in maintenance schema using the eventcode and check for the isEnabled field. If the isEnabled is 'Y' it will store the data in an eventlog table in LMX schema.

We have a cron job that will be triggered in configured time interval which will fetch the value from the integration schema and check for the unpublished message. Those message that are not published and havenot errored out will be published to kafka.

### Maintenance Service Functionality

The oblm-maintenance-services has the following events configured.

1. bank-pref
2. branch-pref
3. pricing-map

The oblm-maintenance-services will check the value of the isEnabled column for the above event\_code, if 'Y' the event will be logged in the lmx schema in lmx\_tb\_event\_log along with the event\_code and event\_topic.

The LMM\_TM\_EVENTS table in the maintenance schema has the following columns

- ID
- EVT\_CODE
- EVT\_CATEGORY
- EVT\_DESC
- EVT\_TOPIC
- EVT\_IENABLED
- MAKER\_ID
- MAKER\_DT\_STAMP
- CHECKER\_ID
- CHECKER\_DT\_STAMP
- RECORD\_STAT
- AUTH\_STAT
- ONCE\_AUTH

- MOD\_NO

Here the event\_code will be predefined by the developers and this event\_code will be used to map an event from service to the even\_topic in which kafka will be publishing.

Depending on the requirement the consumer can alter the value of the isEnabled field to 'Y' if events need to be published for that event.

Events will have been pre-added into the database before the deployment.

### **Sweep Service Functionality**

The oblm-sweep-services has the following events configured.

1. sweep-success (S)
2. sweep-error (E)
3. sweep-pending (P)
4. sweep-handOff (H)

The oblm-sweep-services will call the oblm-maintenance-services and for the above event\_code it will check the value of the isEnabled column, if 'Y' the event will be logged in the lmx schema in lmx\_tb\_event\_log along with the event\_code and event\_topic.

### **Structure Service Functionality**

The oblm-structure-services has the following events configured.

1. structure-created
2. structure-createdAndAuthorized
3. structure-modified
4. structure-modifiedAndAuthorized
5. structure-closed
6. structure-closedAndAuthorized
7. structure-reopen
8. structure-reopenAndAuthorized
9. structure-expiry (structure expiring in n number of days where n is configurable)
10. structure-charge

The oblm-structure-service will call the oblm-maintenance-service and for the above event\_code it will check the value of the isEnabled column, if 'Y' the event will be logged in the lmx schema in lmx\_tb\_event\_log along with the event\_code and event\_topic.

The structure-expiry event is a scheduler. It will be triggered once a day. The scheduler is a cron job, the time is configurable, and it should be cron expression.

Cron expression example: '0 40 20 \* \* ?' will trigger the service endpoint at 8.40pm every day.

### **Integration Service Functionality**

The events that need to be published from the oblm-services will be stored in the lmx\_tb\_events\_log.



The oblm-integration-service has a scheduler that will be triggered in configured interval. The scheduler is a cron job, the time interval is configurable.

The lmx\_tb\_events\_log have columns event is a scheduler. It will be triggered once a day. The scheduler is a cron job, the time is configurable, and it should be cron expression.

Some important columns of lmx\_tb\_events which is generic for all the oblm-services that wants to publish to kafka.

1. ID
2. EVT\_CODE
3. EVT\_TOPIC
4. LOG\_TYPE
5. LOG\_DESCRIPTION
6. LOG\_TIME
7. SERVICE\_DATA
8. PUBLISHED\_TIME
9. IS\_PUBLISHED
10. RETRY\_COUNT
11. EVT\_KEY

**EVT\_TOPIC** is the topic name on which the event will be published

**EVT\_CODE** is unique for each event and it helps to map the events from each service to an event\_topic. The evt\_code is developer specified.

**LOG\_TYPE** is the name of the service which has logged this event in the lmx schema

**LOG\_DESCRIPTION** is the brief description of that particular event.

**SERVICE\_DATA** is the service specific data that will be logged from oblm-services as string

**LOG\_TIME** is the time at which the events from an oblm-service is logged in the lmx schema, or else we can say it is the time at which an event occurred (Example: structure created )

**PUBLISHED\_TIME** is the time at which an event will be published to kafka from oblm-integration-service.

**RETRY\_COUNT** is the number of times an event entry in the lmx\_tb\_event\_log will be retried to send the event for the retryCount(this is configurable will be fetched from properties table, so value of RETRY\_COUNT<= retryCount  publish) number of times, and if it fails for retryCount number of times it will be marked as an error and will not be processed further.

**EVT\_KEY** is the service specific id. If the event is from oblm-sweep-service, it will be storing the sweepId.

**IS\_PUBLISHED** is the column which will store value such a 'Y' if the event is published, 'N' if the event is not published, 'E' if the event couldn't be published for retryCount number of times. Default value of this field will be 'N', which will be updated for the above-mentioned scenarios.

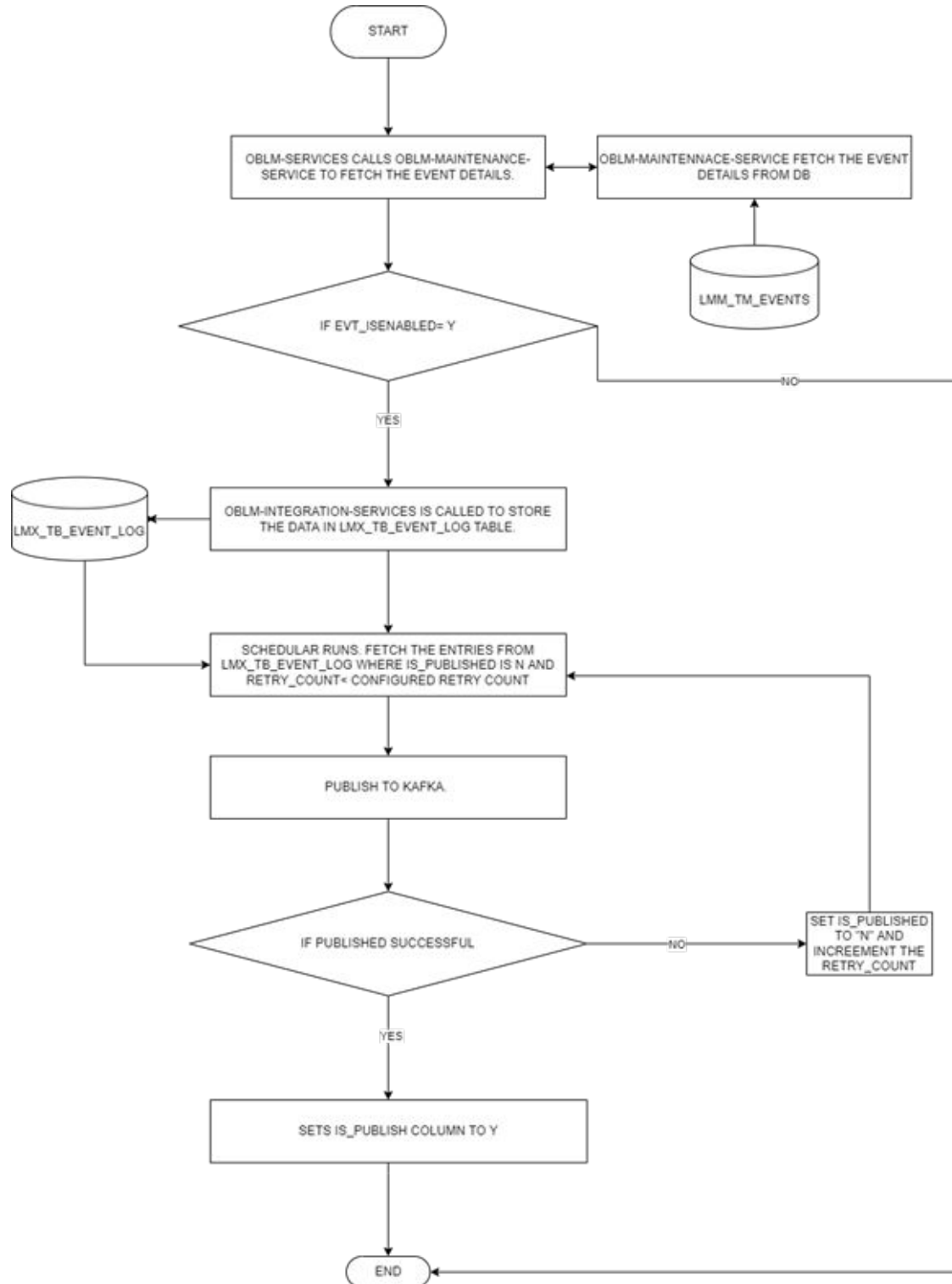
Cron expression example: '0 0/10 \* \* \* ?' will trigger the the service endpoint on every 10 mins

# 7

## Flow Diagram

This topic describes about the flow diagram of Kafka events.

Figure 7-1 Flow Diagram



# 8

## Payload and Header

This topic describes about the various payload and header for Oracle Banking Liquidity Management.

### Generic LM Event Payload

This payload is applicable for the below sweep and structure service events:

- sweep-success (S)
- sweep-error (E)
- sweep-pending (P)
- sweep-handOff (H)
- structure-created
- structure-createdAndAuthorized
- structure-modified
- structure-modifiedAndAuthorized
- structure-closed
- structure-closedAndAuthorized
- structure-reopen
- structure-reopenAndAuthorized
- structure-expiry (structure expiring in n number of days where n is configurable)

#### **Payload:**

id

String,Null

Default: null

evtCode

String,Null

Default: null

logTime

String,Null

Default: null

logType

String,Null

Default: null

logDescription

String,Null

Default: null

serviceData

String,Null

Default: null

publishedTime

String,Null

Default: null

### **Bank Preference Event Payload**

#### **Payload:**

id

String,Null

Default: null

modNo

String,Null

Default: null

RecordStat

String,Null

Default: null

AuthStat

String,Null

Default: null

MakerId

String,Null

Default: null

MakerDateStamp

String,Null

Default: null

CheckerId

String,Null

Default: null

checkerDateStamp

String,Null

Default: null

---

OnceAuth  
String,Null  
Default: null

applicationCode  
String,Null  
Default: null

bankCode  
String,Null  
Default: null

chargeCalcPref  
String,Null  
Default: null

chargeCollPref  
String,Null  
Default: null

chgIncludeClosedVa  
String,Null  
Default: null

### **Branch Preference Event Payload**

#### **Payload:**

id  
String,Null  
Default: null

modNo  
String,Null  
Default: null

RecordStat  
String,Null  
Default: null

AuthStat  
String,Null  
Default: null

MakerId  
String,Null

Default: null

MakerDateStamp

String,Null

Default: null

CheckerId

String,Null

Default: null

checkerDateStamp

String,Null

Default: null

OnceAuth

String,Null

Default: null

applicationCode

String,Null

Default: null

branchCode

String,Null

Default: null

chargeRateCode

String,Null

Default: null

chargeRateType

String,Null

Default: null

### **Structure Charge Event Payload**

#### **Payload:**

id

String,Null

Default: null

modNo

String,Null

Default: null

RecordStat

---

String,Null  
Default: null  
AuthStat  
String,Null  
Default: null  
MakerId  
String,Null  
Default: null  
MakerDateStamp  
String,Null  
Default: null  
CheckerId  
String,Null  
Default: null  
CheckerDateStamp  
String,Null  
Default: null  
OnceAuth  
String,Null  
Default: null  
applicationCode  
String,Null  
Default: null  
strCode  
String,Null  
Default: null  
realCustomerNo  
String,Null  
Default: null  
chgFundingAccount  
String,Null  
Default: null  
chgFundingAccountBranch  
String,Null



Default: null

chgFundingAccountCCY

String,Null

Default: null

vaCount

String,Null

Default: null

event

String,Null

Default: null

strChgType

String,Null

Default: null

### **Pricing Map Event Payload**

#### **Payload:**

id

String,Null

Default: null

modNo

String,Null

Default: null

RecordStat

String,Null

Default: null

AuthStat

String,Null

Default: null

MakerId

String,Null

Default: null

MakerDateStamp

String,Null

Default: null

CheckerId

String,Null

Default: null

checkerDateStamp

String,Null

Default: null

OnceAuth

String,Null

Default: null

applicationCode

String,Null

Default: null

pricingScheme

String,Null

Default: null

realCustomerNo

String,Null

Default: null

chgFundingAccount

String,Null

Default: null

chgFundingAccountBranch

String,Null

Default: null

chgFundingAccountCCY

String,Null

Default: null

chgPostingBranch

String,Null

Default: null

event

String,Null

Default: null

## Header

### Common Header:

userId  
String  
branchCode  
String  
sourceSystem  
String  
event  
String  
ackRequired  
Boolean  
Default: false  
kafka\_messageKey  
String  
messageId  
String  
entityId  
String

# 9

## Tables

### LMM\_TM\_EVENTS

In the below table, we configure all the events that Oracle Banking Liquidity Management is supporting. Here we can toggle the evt\_isEnabled column to “Y” (If we want to publish that event) or “N”(If we want to publish that event).

**Table 9-1 LM\_TM\_EVENTS Table**

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID	VARCHAR2 (36 BYTE)	No	(null)	1	(null)
EVT_CODE	VARCHAR2 (50 BYTE)	No	(null)	2	(null)
EVT_CATEGORY	VARCHAR2 (20 BYTE)	No	(null)	3	(null)
EVT_DESC	VARCHAR2 (100 BYTE)	No	(null)	4	(null)
EVT_TOPIC	VARCHAR2 (50 BYTE)	Yes	(null)	5	(null)
EVT_IENABLED	CHAR (1 BYTE)	No	(null)	6	(null)
MAKER_ID	VARCHAR2 (12 BYTE)	Yes	(null)	7	(null)
MAKER_DT_STAMP	DATE	Yes	(null)	8	(null)
CHEKER_ID	VARCHAR2 (12 BYTE)	Yes	(null)	9	(null)
CHECKER_DT_STAMP	DATE	Yes	(null)	10	(null)
RECORD_STAT	CHAR (1 BYTE)	Yes	(null)	11	(null)
AUTH_STAT	CHAR (1 BYTE)	Yes	(null)	12	(null)
ONCE_AUTH	CHAR (1 BYTE)	Yes	(null)	13	(null)
MOD_NO	NUMBER (4,0)	Yes	(null)	14	(null)

### LMX\_TB\_EVENT\_LOG

In the below table, all the Oracle Banking Liquidity Management services that wants to publish will store their payload and a scheduler will fetch data from this table and fetch all the records where is\_published is “N” and retry\_count<=max\_retry\_configured.

**Table 9-2 LMX\_TB\_EVENT\_LOG Table**

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID	VARCHAR2 (36 BYTE)	No	(null)	1	(null)

**Table 9-2 (Cont.) LMX\_TB\_EVENT\_LOG Table**

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
EVT_CODE	VARCHAR2 (50 BYTE)	No	(null)	2	(null)
EVT_TOPIC	VARCHAR2 (50 BYTE)	No	(null)	3	(null)
EVT_KEY	VARCHAR2 (50 BYTE)	Yes	(null)	4	(null)
LOG_TYPE	VARCHAR2 (20 BYTE)	Yes	(null)	5	(null)
LOG_DESCRIPTION	VARCHAR2 (500 BYTE)	Yes	(null)	6	(null)
LOG_TIME	TIMESTAMP (6)	Yes	(null)	7	(null)
SERVICE_DATA	CLOB	Yes	(null)	8	(null)
PUBLISHED_TIME	TIMESTAMP (6)	Yes	(null)	9	(null)
IS_PUBLISHED	CHAR (1 BYTE)	Yes	'N'	10	(null)
RETRY_COUNT	NUMBER	Yes	0	11	(null)

**PLATO\_EVENTHUB\_OUT\_LOG**

The below table is provided in by the plato-event-hub-core (in LMX schema in Oracle Banking Liquidity Management). Here all the events that are to be published are stored along with the publisher service name and status is changed to success once successfully published to kafka.

**Table 9-3 LMX\_TB\_EVENT\_LOG Table**

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID	VARCHAR2 (36 BYTE)	No	(null)	1	(null)
TOPIC_NAME	VARCHAR2 (255 BYTE)	No	(null)	2	(null)
MESSAGE_KEY	VARCHAR2 (36 BYTE)	Yes	(null)	3	(null)
EVENT_TYPE	VARCHAR2 (25 BYTE)	Yes	(null)	4	(null)
PAYLOAD	CLOB	Yes	(null)	5	(null)
EXCEPTION	VARCHAR2 (512 BYTE)	Yes	(null)	6	(null)
STATUS	VARCHAR2 (33 BYTE)	Yes	(null)	7	(null)
RETRY_COUNT	NUMBER	Yes	(null)	8	(null)
RETRY_DATETIME	DATE	Yes	(null)	9	(null)
CREATED_BY	VARCHAR2 (12 BYTE)	Yes	(null)	10	(null)
CREATED_DATE	DATE	Yes	(null)	11	(null)
UPDATED_BY	VARCHAR2 (12 BYTE)	Yes	(null)	12	(null)
UPDATED_DATE	DATE	Yes	(null)	13	(null)

**Table 9-3 (Cont.) LMX\_TB\_EVENT\_LOG Table**

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
CORRELATION_ID	VARCHAR2 (256 BYTE)	Yes	(null)	14	(null)
APPLICATION_NAME	VARCHAR2 (120 BYTE)	Yes	(null)	15	(null)
ACK_COUNT	NUMBER (38, 0)	Yes	0	16	(null)
HEADER	CLOB	Yes	(null)	17	(null)
CONSUMER_APP L...	VARCHAR2 (512 BYTE)	Yes	(null)	18	(null)

**PLATO\_EVENTHUB\_IN\_LOG**

The below table is provided in by the plato-event-hub-core (in LMX schema in Oracle Banking Liquidity Management). Here all the events that are consumed are stored along with the consumer service name.

**Table 9-4 LMX\_TB\_EVENT\_LOG Table**

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID	VARCHAR2 (36 BYTE)	No	(null)	1	(null)
TOPIC_NAME	VARCHAR2 (100 BYTE)	Yes	(null)	2	(null)
MESSAGE_KEY	VARCHAR2 (255 BYTE)	Yes	(null)	3	(null)
EVENT_TYPE	VARCHAR2 (36 BYTE)	Yes	(null)	4	(null)
EVENT_PAYLOAD	CLOB	Yes	(null)	5	(null)
STATUS	VARCHAR2 (36 BYTE)	Yes	(null)	6	(null)
EXCEPTION	VARCHAR2 (500 BYTE)	Yes	(null)	7	(null)
MSG_DT_STAMP	DATE	Yes	(null)	8	(null)
CORRELATION_ID	VARCHAR2 (256 BYTE)	Yes	(null)	9	(null)
APPLICATION_NAME	VARCHAR2 (100 BYTE)	Yes	(null)	10	(null)

# Index

## C

---

Changes to Clients, [5-6](#)

Configure Brokers, [5-5](#)

Create Users in Zookeeper, [5-4](#)

## E

---

Export Private Key as Certificate, [5-3](#)

## G

---

Generate Keystore, [5-1](#)

## I

---

Import the Certificate and Generate Trust Store,  
[5-3](#)

Important Commands, [3-1](#)

Integration Service Functionality, [6-2](#)

## K

---

Kafka Middleware Setup, [2-1](#)

Kafka Setup, [2-2](#)

## M

---

Maintenance Service Functionality, [6-1](#)

## P

---

Payload and Header, [8-1](#)

## S

---

Security - SSL Encryption with SASL-SCRAM  
Authentication, [5-1](#)

Structure Service Functionality, [6-2](#)

Sweep Service Functionality, [6-2](#)

## Z

---

Zookeeper Setup, [2-1](#)