

**Kafka Configuration Guide**

# **Oracle Banking Liquidity Management**

Release 14.5.4.0.0

**Part Number F54228-01**

February 2022

## Kafka Configuration Guide

Oracle Financial Services Software Limited  
Oracle Park  
Off Western Express Highway  
Goregaon (East)  
Mumbai, Maharashtra 400 063  
India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax: +91 22 6718 3001

<https://www.oracle.com/industries/financial-services/index.html>

Copyright © 2018, 2022, Oracle and/or its affiliates. All rights reserved.

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 failsafe, 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.

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.

This software or hardware and documentation may provide access to or information on 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. 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.

# Contents

1	Preface .....	1
1.1	Purpose .....	1
1.2	Audience .....	1
1.3	Related Documents .....	1
2	Prerequisites .....	2
3	Kafka Middleware Setup .....	3
3.1	Zookeeper Setup .....	3
3.2	Kafka Setup .....	5
4	Important Commands .....	7
4.1	View the Topic Configurations .....	7
4.2	View the Messages Sent from producer-consumer .....	7
4.3	Creating Kafka Topics manually .....	7
5	Increasing Replication Factor for an existing topic .....	8
6	Security- SSL Encryption with SASL-SCRAM authentication .....	10
6.1	Generate Keystore .....	10
6.2	Export Private Key as Certificate .....	12
6.3	Import the Certificate and Generate Trust Store .....	12
6.4	Creation of Users in Zookeeper .....	13
6.5	Configuring Brokers .....	13
6.6	Changes to clients .....	14
6.7	Important commands .....	16
7	Implementation .....	18
7.1	The Flow .....	18
7.2	Maintenance Service Functionality .....	18
7.3	Sweep Service Functionality .....	19
7.4	Structure Service Functionality .....	19
7.5	Integration Service Functionality .....	20
8	Flow Diagram .....	22
9	Payload and Header .....	23
9.1	Generic LM Event Payload (for sweep and structure service events) .....	23
9.2	Bank Preference Event Payload .....	24
9.3	Branch Preference Event Payload .....	25
9.4	Structure Charge Event Payload .....	26
9.5	Pricing Map Event Payload .....	28
9.6	Header .....	29

10	Tables.....	30
10.1	LMM_TM_EVENTS.....	30
10.2	LMX_TB_EVENT_LOG.....	30
10.3	PLATO_EVENTHUB_OUT_LOG: .....	31
10.4	PLATO_EVENTHUB_IN_LOG:.....	31

# **1 Preface**

## **1.1 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.

## **1.2 Audience**

This guide is intended for the implementation teams.

## **1.3 Related Documents**

The related documents are as follows:

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

## 2 Prerequisites

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

This guide describes about the above configuration. These values can be increased based on the requirement and load. Restrict the access to the server\*.properties file of Kafka servers.

## 3 Kafka Middleware Setup

### 3.1 Zookeeper Setup

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. Download Zookeeper and unzip it.

Consider that zookeeper is extracted in /tools/zookeeper on both the servers.

2. Copy and rename **zoo\_sample.cfg** to **zookeeper1.cfg** in /tools/zookeeper/conf on server 1.
3. Open **zookeeper1.cfg** and add/modify the below properties using any text editor.

```
dataDir= <zookeeper 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
```

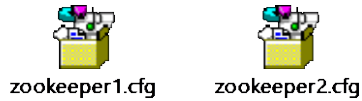
**NOTE:** Update the IP value with the respective server IP.

4. Copy the **zoo.cfg** file and paste in the same directory with new name **zookeeper2.cfg** on Server 2 (Other names can also be used). These configuration files used for each of the zookeeper nodes.

5. Open **zookeeper2.cfg** and modify the property,

```
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.



**NOTE:** Refer PDF Attachments to download the above attachments.

6. 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.
7. 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.
8. 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
```



## 3.2 Kafka Setup

1. Download Kafka and unzip it.

Consider that the kafka is extracted in /tools/kafka on both the servers.

2. Navigate to config folder in Apache Kafka (/tools/kafka/config), rename **server.properties** to **server1.properties** and add/modify the below 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.segment.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>,<zook
eeper_hostname_2>:<zookeeper_client_port>,<zookeeper_hostname_3>:<zook
```

### 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 your 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.

3. Take the duplicate of **server1.properties** and paste into the same directory with name **server2.properties** on server 2.

4. Open **server2.properties** and modify the below,

```
broker.id=1
port=9093
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.



**NOTE:** Refer PDF Attachments to download the above attachments.

5. To run Kafka brokers, change path to /tools/kafka directory and run 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
```

6. 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.
7. At present, kafka takes the default value for message size as:

```
message.max.bytes=1000012
```

8. Add and update this field in server\*.properties for increasing based on requirement.
9. 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

## 4 Important Commands

### 4.1 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
```

### 4.2 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
```

### 4.3 Creating 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.

## 5 Increasing 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. Create a json file as below in the config path.



increase-replication-f  
actor.json

**NOTE:** Refer PDF Attachments to download the above attachment.

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":"stru  
cture-closed","partition":0,"replicas":[1],"log_dirs":["any"]}]}
```

Save this to use as the --reassignment-json-file option during rollback

Successfully started reassignment of partitions.

3. 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

## 4. 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

## 6 Security- SSL Encryption with SASL-SCRAM authentication

### 6.1 Generate Keystore

The items highlighted in blue 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
```

In the above command,

1. **alias** is used to identify the public and private key pair created.
2. **keyalg** is the key algorithm used to generate the public and private key pair. The RSA key algorithm is recommended.
3. **keysize** 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.
4. **sigalg** 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.
5. **valdays** is the number of days for which the certificate is to be considered valid. Please consult with your CA on this period.
6. **keystore** 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 will prompt for the following attributes of the certificate and keystore:

1. **Keystore Password:** Specify a password that will be used to access the keystore. This password needs to be specified later, when configuring the identity store in Kafka Server.
2. **Key Password:** Specify a password that will be 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.
3. **First and Last Name (CN):** Enter the domain name of the machine used to access Oracle Banking Liquidity Management, for instance, www.example.com
4. **Name of your Organizational Unit:** The name of the department or unit making the request, for example, Oracle Banking Liquidity Management. Use this field to further identify the SSL Certificate you are creating, for example, by department or by physical server.

5. **Name of your Organization:** 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.
6. **Name of your City or Locality:** The city in which your organization is physically located, for example Bengaluru.
7. **Name of your State or Province:** The state/province in which your organization is physically located, for example Karnataka.
8. **Two-letter Country Code for this Unit:** The country in which your organization is physically located, for example US, UK, IN etc.

**Example:**

Listed below is the result of a sample execution of the command:

```
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>

## 6.2 Export Private Key as Certificate

```
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>

## 6.3 Import the Certificate and Generate Trust Store

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

In the above command:

1. **alias** 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.
2. **cert\_file** is the location of the file containing the PKCS#7 formatted reply from the CA, containing the signed certificate.
3. **truststore** is the location where the truststore should be generated.
4. **storepass** is the password for the truststore.

Generate two trust stores from the same certificate. One used for kafka server and one 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 would be needed for this method:

1. KafkaServerKeystore.jks : keystore file for Kafka brokers
2. KafkaServerTrustStore.jks : Truststore file for server
3. KafkaClientTrustStore.jks : Truststore file for client.

The KafkaClientTrustStore.jks file need to be imported by every client to validate the server.

**NOTE:** The truststore files should be generated using the same CA. Generate and place these files on all the different servers of kafka so that it can be accessed by server\*.properties file. The KafkaClientTrustStore.jks should be placed on the server, which is accessible by the microservices also.



## 6.4 Creation of Users in Zookeeper

Start the zookeeper (command in section 2.1) and execute the below commands for the 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
```

```
./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
```

Two users are created above with alice and admin as usernames and two different passwords are setup for each user one for each scram mechanism. Here, the user 'admin' is used for Kafka broker auth and 'alice' is used for client auth. For multiple zookeeper nodes, use comma separated serverIP:port like in the above example(localhost:2181,localhost:2182).

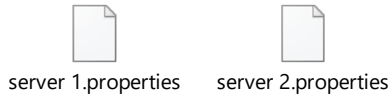
## 6.5 Configuring Brokers

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:9092
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 file is attached below:



**NOTE:** Refer PDF Attachments to download the above attachments.

Start the kafka servers (refer command in Section 3.2)

## 6.6 Changes to clients

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

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\securityKeys\KafkaClientTrustStore.jks');
```

```

insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10114,'oblm-structure-
services','jdbc','jdbc','plato.services.kafka.truststore.password','{cipher}7ec1250634259a1af12f74a7e4705ade7493a4695cc1efd3b713571453fda266');

insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10115,'oblm-structure-
services','jdbc','jdbc','spring.cloud.stream.kafka.binder.configuration.s
asl.mechanism','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.loginModul
e','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.us
ername','alice');

insert into PROPERTIES (ID,APPLICATION,PROFILE,LABEL,KEY,VALUE) values
(10118,'oblm-structure-
services','jdbc','jdbc','spring.cloud.stream.kafka.binder.jaas.options.pa
ssword','{cipher}2f32dc1770acec085105e3ba585cc44c71534451b88b6047504f1119
1ad8cc1f');

```

## 6.7 Important commands

Creation of topics manually is same as the command mentioned in section 3.3. But if you 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



ssl.properties

**NOTE:** Refer PDF Attachments to download the above attachments.

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
```

## 7 Implementation

### 7.1 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.

### 7.2 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.

### 7.3 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.

### 7.4 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.

## 7.5 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 (eg : structure created )



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

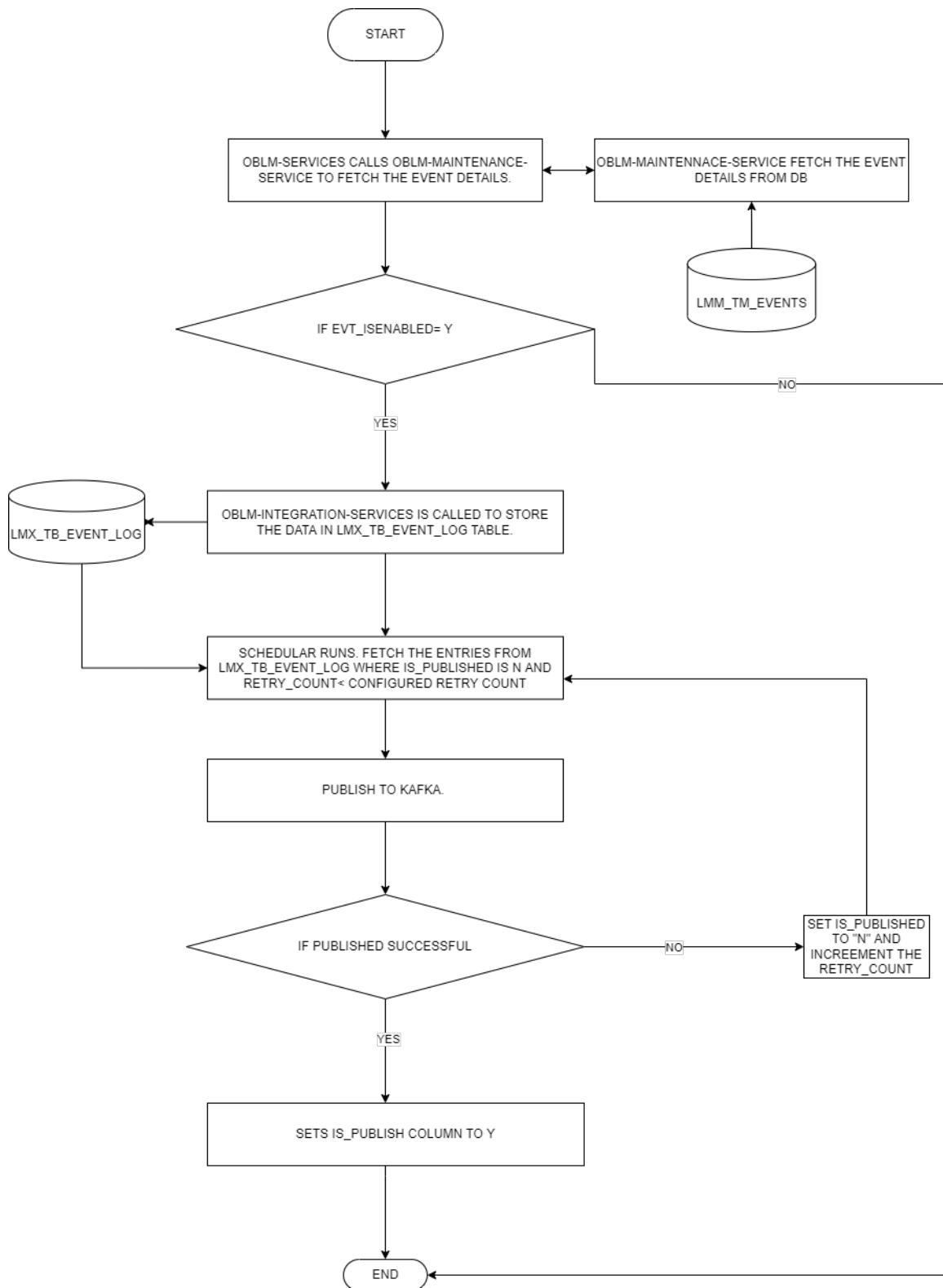
**RETRY\_COUNT** is the number of times an event entry in the lmx\_tb\_event\_log will be retried to send the event for the retryCount(this is configurable will be fetched from properties table, so value of RETRY\_COUNT<= retryCount □ publish) number of times, and if it fails for retryCount number of 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

## 8 Flow Diagram



## 9 Payload and Header

### 9.1 Generic LM Event Payload (for sweep and structure service events)

This payload is applicable for the below 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

## 9.2 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
```

### 9.3 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

```

## 9.4 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

## 9.5 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

```

## 9.6 Header

### Common Header:

```

userId
String
branchCode
String
sourceSystem
String
event
String
ackRequired
Boolean
Default: false
kafka_messageKey
String
messageId
String
entityId
String

```

## 10 Tables

### 10.1 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).

❖ 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)	
CHECKER_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)	

### 10.2 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.

❖ 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_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 B...	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)	

### 10.3 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.

❖ COLUMN_NAME	❖ DATA_TYPE	❖ NULLABLE	DATA_DEFAULT	❖ COLUMN_ID	❖ COMMENTS
ID	VARCHAR2(36 BYTE)	No	(null)	1	(null)
TOPIC_NAME	VARCHAR2(255 B...	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 B...	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)
CORRELATION_ID	VARCHAR2(256 B...	Yes	(null)	14	(null)
APPLICATION_NAME	VARCHAR2(120 B...	Yes	(null)	15	(null)
ACK_COUNT	NUMBER(38,0)	Yes	0	16	(null)
HEADER	CLOB	Yes	(null)	17	(null)
CONSUMER_APPL...	VARCHAR2(512 B...	Yes	(null)	18	(null)

### 10.4 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.

❖ COLUMN_NAME	❖ DATA_TYPE	❖ NULLABLE	DATA_DEFAULT	❖ COLUMN_ID	❖ COMMENTS
ID	VARCHAR2(36 BYTE)	No	(null)	1	(null)
TOPIC_NAME	VARCHAR2(100 B...	Yes	(null)	2	(null)
MESSAGE_KEY	VARCHAR2(255 B...	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 B...	Yes	(null)	7	(null)
MSG_DT_STAMP	DATE	Yes	(null)	8	(null)
CORRELATION_ID	VARCHAR2(256 B...	Yes	(null)	9	(null)
APPLICATION_NAME	VARCHAR2(100 B...	Yes	(null)	10	(null)