

# **Data Transfer Service**

## **Data Subscription (Kafka Edition)**

### **Product Documentation**



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Data Subscription (Kafka Edition)

### Databases Supported by Data Subscription

### MySQL series Data Subscription

#### Creating MySQL or TDSQL for MySQL Data Subscription

#### Creating TDSQL for MySQL Data Subscription

#### Creating MariaDB/Percona Data Subscription

### Consuming MySQL Data

#### Operation Guide for Consuming MySQL Series Subscribed Data

#### Demo Description

##### ProtoBuf Demo Description

##### Avro Demo Description

##### JSON Demo Description

### Consuming Data with Flink

#### Consuming MySQL or TDSQL-C for MySQL Subscribed Data Using Flink

#### Consuming TDSQL for MySQL Data with Flink

#### Demo Description

##### Avro Demo Description (Flink)

##### ProtoBuf Demo Description (Flink)

### Advanced Subscription Operations

#### Setting Partitioning Policy

### Fix for Verification Failure

#### Database Connection Check

#### Peripheral Check

#### Version Check

#### Source Instance Permission Check

#### Binlog Parameter Check

## Data Subscription for TDSQL PostgreSQL

### Creating Data Subscription for TDSQL PostgreSQL Edition

### Consuming TDSQL for PostgreSQL Data

### Fix for Verification Failure

#### Database Connection Check

#### Version Check

#### Peripheral Check

#### Source Instance Permission Check

#### Constraint Check

## MongoDB Data Subscription

- Creating Data Subscription for MongoDB

- Consuming MongoDB Data

## Task Management

- Task Status Description

- Viewing Subscription Details

- Modifying Subscribed Object

- Pay-as-You-Go to Monthly Subscription

- Resetting Subscription

- Terminating/Returning an Instance

## Consumption Management

- Creating Consumer Group

- Managing Consumer Group

- Modify Consumption Site offset

# Data Subscription (Kafka Edition)

## Databases Supported by Data Subscription

Last updated : 2024-07-08 16:45:54

Data subscription refers to the process where DTS gets the data change information of a key business in the database, converts it into message objects, and pushes them to Kafka for the downstream businesses to subscribe to, obtain, and consume. DTS allows you to directly consume data through a Kafka/Flink client, so that you can implement data sync between TencentDB databases and heterogeneous systems, such as cache update, real-time ETL (a data warehousing technology) sync, and async business decoupling.

Subscribed data in **Protobuf**, **Avro**, or **JSON** formats can be consumed. ProtoBuf and Avro adopt the binary format with higher consumption efficiency, while JSON adopts the easier-to-use lightweight text format.

DTS supports data subscription for the following types of databases.

Source Database Type	Source Database Version	Subscribable Data Type	Format of Subscribed Data	Format of Data Subscribed by Flink (DataStream API)
MySQL	Self-built MySQL 5.5, 5.6, 5.7, 8.0 TencentDB for MySQL 5.5, 5.6, 5.7, 8.0	Data update Structure update Full instance	Protobuf/Avro/JSON	Avro
MariaDB	Self-built MariaDB 5.5, 10.0, 10.1 TencentDB for MariaDB (kernel version: Percona 5.7, MySQL 8.0, and MariaDB 10.1)	Data update Structure update Full instance	Protobuf	Not supported
Percona	Self-built Percona 5.5, 5.6, 5.7, 8.0	Data update Structure update Full instance	Protobuf	Not supported
TDSQL for MySQL	TDSQL for MySQL (kernel version :MySQL 8.0 and Percona 5.7)	Data update Structure update Full instance	Protobuf	Protobuf
TDSQL-C for MySQL	TDSQL-C for MySQL 5.7, 8.0	Data update	Protobuf/Avro/JSON	Avro

		Structure update Full instance		
TDSQL for PostgreSQL	TDSQL for PostgreSQL	Data update	Protobuf	Not supported
MongoDB	TencentDB for MongoDB 3.6, 4.0, 4.2, 4.4	Data update	JSON	Not supported

# MySQL series Data Subscription

## Creating MySQL or TDSQL for MySQL Data Subscription

Last updated : 2024-07-08 16:52:02

This document describes how to create a data subscription task in DTS for TencentDB for MySQL or TDSQL-C for MySQL.

### Prerequisites

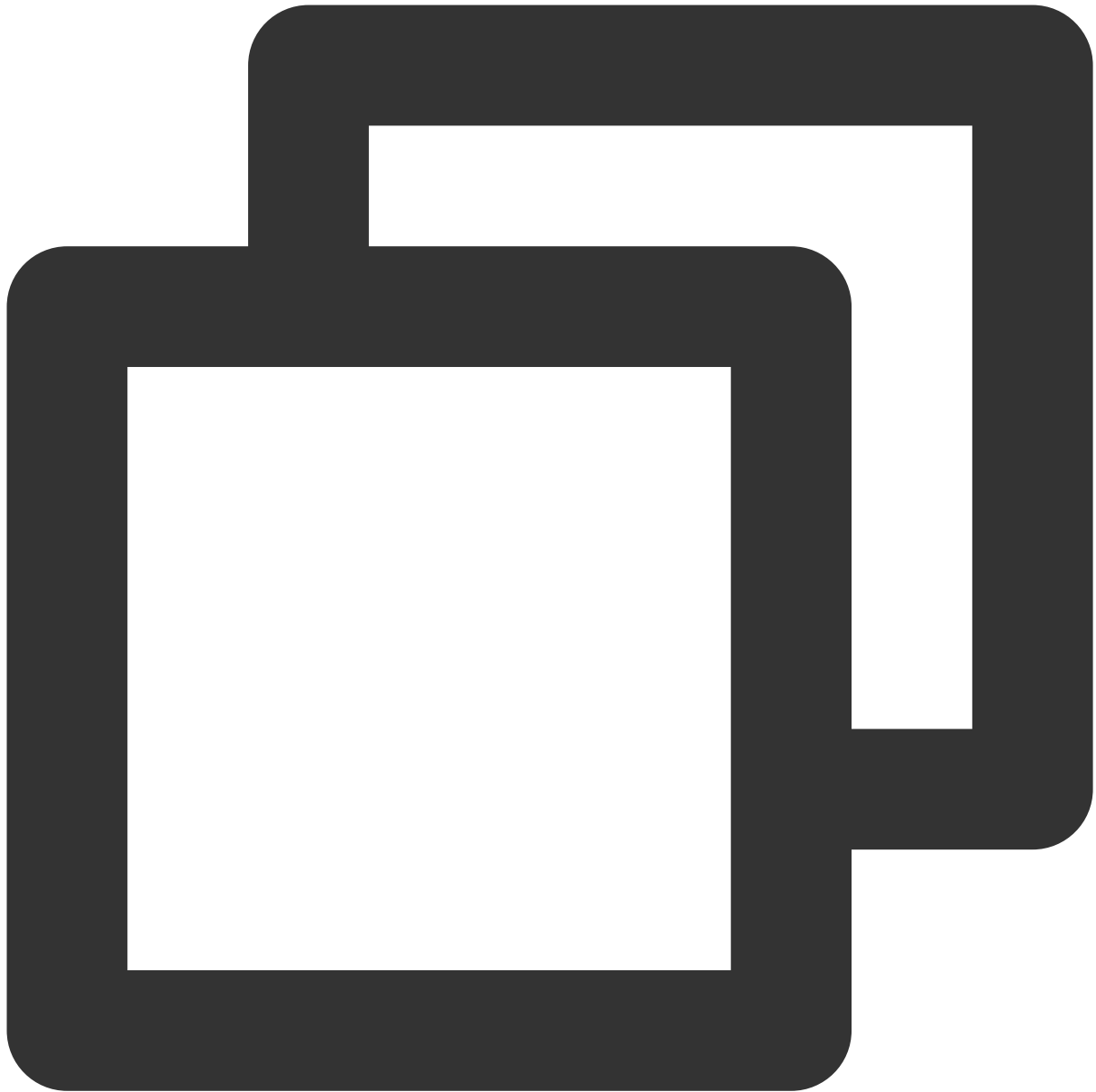
You have prepared a TencentDB instance to be subscribed to, and the database version meets the requirements. For more information, see [Databases Supported by Data Subscription](#).

You have enabled the binlog in the source instance.

You have created a subscription account in the source instance and granted it the following permissions:

REPLICATION CLIENT, REPLICATION SLAVE, PROCESS, and SELECT for all objects.

Authorization statements are as follows:



```
create user 'migration account' IDENTIFIED BY 'account password';  
grant SELECT, REPLICATION CLIENT, REPLICATION SLAVE, PROCESS on *.* to 'migration acc  
flush privileges;
```

## Restrictions

Currently, the subscribed message content is retained for 1 day by default. Once expired, the data will be cleared. Therefore, you need to consume the data promptly.



The region where the data is consumed should be the same as that of the subscribed instance.

Geometry data types are not supported currently.

The delivery semantics of subscription to Kafka messages in DTS is to deliver a message at least once; therefore, the consumed data may be duplicated in special cases. For example, if the subscription task is restarted, the source binlog will be pulled before the interruption offset after the restart, resulting in repeated delivery of messages.

Operations in the console such as modifying subscribed objects and restoring abnormal tasks may cause duplicate messages. If your business is sensitive to duplicate data, you need to add deduplication logic based on the business data in the consumption demo.

## SQL operations for subscription

Operation Type	Supported SQL Operations
DML	INSERT, UPDATE, and DELETE
DDL	CREATE DATABASE, DROP DATABASE, CREATE TABLE, ALTER TABLE, DROP TABLE, and RENAME TABLE

## Directions

1. Log in to the [DTS console](#), select **Data Subscription** on the left sidebar, and click **Create Subscription**.

2. On the **Create Subscription** page, select appropriate configuration items and click **Buy Now**.

Billing Mode: Monthly subscription and pay-as-you-go billing are supported.

Region: The region must be the same as that of the database instance to be subscribed to.

Database: Select your actual database type.

Version: Select **Kafka Edition**. You can directly consume data on a Kafka client.

Subscription Name: Edit the name of the current data subscription instance.

3. After successful purchase, return to the data subscription list. You need to click **Configure Subscription** in the **Operation** column to configure the newly purchased subscription before you can use it.

4. On the **Subscription Configuration** page, select the appropriate configuration items and click **Next**.

Instance: Select a database instance. Currently, read-only and disaster recovery instances do not support data subscription.

Database Account: Add the account and password of the instance to be subscribed to. The account must have the permissions required by the subscription task, including REPLICATION CLIENT, REPLICATION SLAVE, PROCESS, and SELECT of all objects.

**Number of Kafka Partitions:** Set the number of Kafka partitions. Increasing the number can improve the speed of data write and consumption. A single partition can guarantee the order of messages, while multiple partitions cannot. If you have strict requirements for the order of messages during consumption, set this value to 1.

**1 Select Instance** > **2 Subscription Type and Object** > **3 Pre-verification**

**Data Subscription Task Setting**

Subscription ID / Name: subs-1 (nar)

Instance Type: MySQL

Instance Name: cdb- Supported instances include: two-node and three-node TencentDB for MySQL 5.5/5.6/5.7/8.0 instances, as well as DR instances.

Database Account:  The database account cannot be empty.

Password:  Database password cannot be empty.

Number of Kafka Partitions: 1 **4** 8

**Next**

5. On the **Subscription Type and Object** page, select a subscription type and click **Save**.

**Subscription Type:** Options include **Data Update**, **Structure Update**, and **Full**.

**Data Update:** Data updates of the selected objects are subscribed to, including INSERT, UPDATE, and DELETE operations.

**Structure Update:** Creation, modification, and deletion of the structures of all objects in the instance are subscribed to.

**Full:** Data and structure updates of all objects in the instance are subscribed to.

**Format of Subscribed Data:** You can select **ProtoBuf**, **Avro**, or **JSON**. ProtoBuf and Avro adopt the binary format with a higher consumption efficiency, while JSON adopts the easier-to-use lightweight text format.

**Kafka Partitioning Policy:** Select **By table name** or **By table name + primary key**.

**Custom Partitioning Policy:** Customize partitions as needed. For more information, see [Setting Partitioning Policy](#).

✓ Select Instance

2 Subscription Type and Object

3 Pre-verification

Subscription ID / Name

subs- (name- )

MySQL Instance

cdb- (dts- )

Subscription Type

☒ Data Update
 ☒ Structure Update
 ☒ Full

Structure update includes the structure creation, deletion, and modification of all objects subscribed to the entire instance.

Format of Subscribed Data

ProtoBuf
  Avro
  JSON

ProtoBuf and Avro are more efficient binary formats, while JSON is an easier-to-use lightweight text format.

Kafka Partitioning Policy

By table name
  By table name + primary key

Custom Partitioning Policy

☒

Custom Partitioning Policy

Custom partitioning rules are applied to objects that meet the following database/table rules. These rules must be described according to the RE2 syntax. For details, see [Syntax](#).

Partitioning policy description:

By table name: Data tables will be matched if their database and table names are described in the regexes below. Messages with the same table name are always put in the same partition, and data changes in the same table are always obtained sequentially.

By table name + primary key: Data tables will be matched if their database and table names are described in the regexes below. Data is partitioned by the combination of table name and primary key, and data in the same table is partitioned by primary key.

By column: Data tables will be matched if their database and table names are described in the regexes below. Data (including data in the same table) is partitioned by column name.

Database Name Match	Table Name Match	Partitioning Policy	Custom Partition Column	Operation
<input type="text" value="^A\$"/> ✓	<input type="text" value="^test\$"/> ✓	By column ▼	<input type="text" value="class"/> ✓	Delete

Add

Strategy Combo Result

When you enable the custom partitioning policy option, your custom partitioning policies will be applied first, followed by the Kafka partitioning policies.

The data matched by the regexes "^A\$" and "^test\$" will be partitioned by column "class" and then routed to Kafka partitions.

The data in a table that cannot be partitioned using the custom partitioning policies will be routed to Kafka partitions by default policy (By table name).

Previous

Save

6. On the **Pre-verification** page, a pre-verification task will run for 2–3 minutes. After the pre-verification is passed, click **Start** to complete data subscription task configuration.

#### Note:

If the verification fails, fix the problem as instructed in [Database Connection Check](#) and initiate the verification again.

7. After you click **Start**, the subscription task will be initialized, which will take 3–4 minutes. After successful initialization, the task will enter the **Running** status.

8. [Add a consumer group](#). Data subscription (Kafka Edition) allows you to create multiple consumer groups for multi-point consumption. The consumption depends on the consumer groups of Kafka; therefore, you must create a consumer group first before data can be consumed.

9. After the subscription instance enters the **Running** status, you can start consuming data. For consumption in Kafka, you need to verify the password. We provide demo code in multiple programming languages and descriptions of main consumption processes and key data structures.

# Creating TDSQL for MySQL Data Subscription

Last updated : 2024-07-08 16:52:02

This document describes how to create a data subscription task in DTS for TDSQL for MySQL.

## Prerequisite

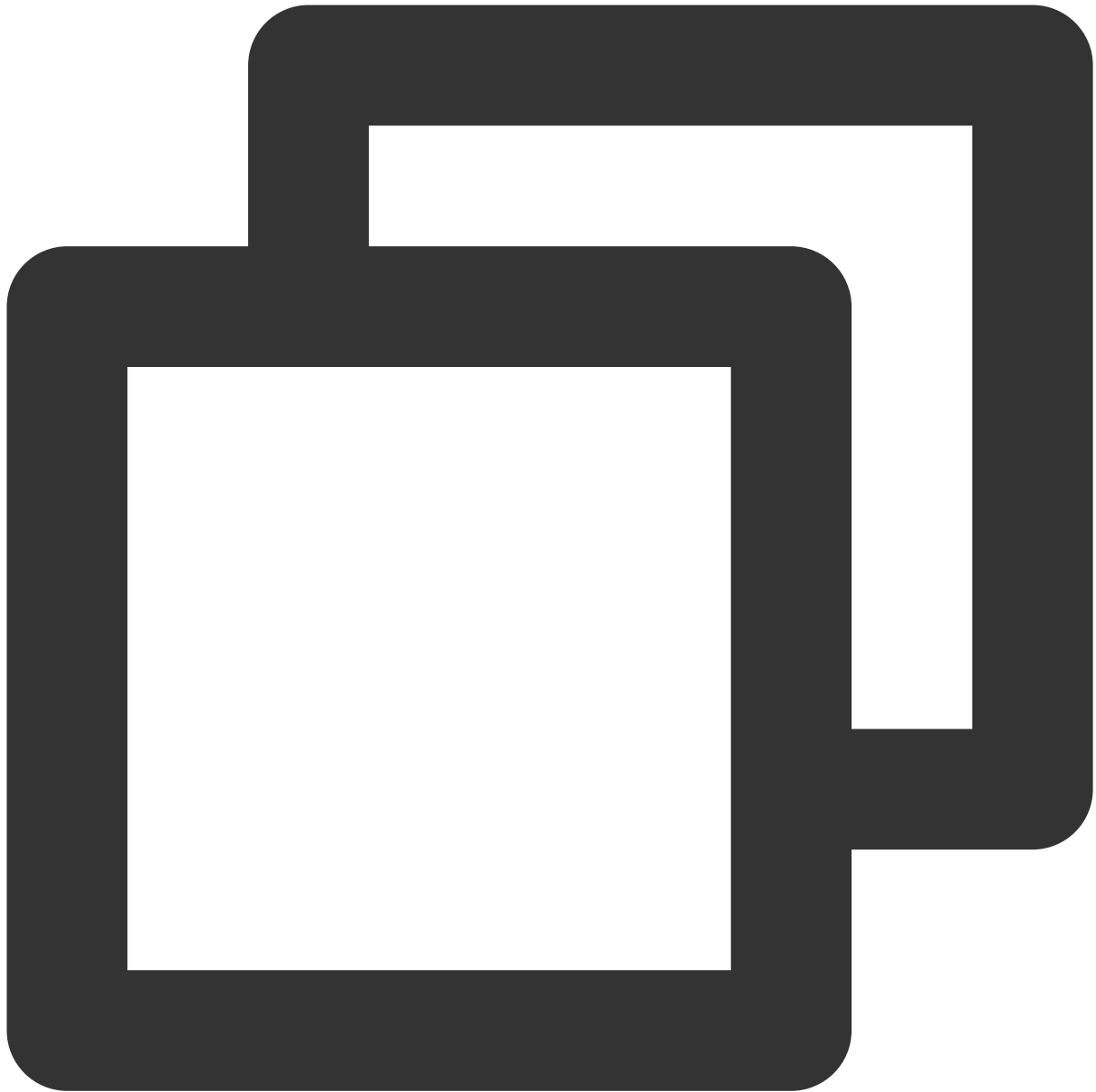
You have prepared a TencentDB instance to be subscribed to, and the database version meets the requirements. For more information, see [Databases Supported by Data Subscription](#).

You have enabled the binlog in the source database.

You have created a subscription account in the source database and granted it the following permissions:

REPLICATION CLIENT, REPLICATION SLAVE, PROCESS, and SELECT for all objects.

Below is the specific authorization syntax:



```
create user 'migration account' IDENTIFIED BY 'account password';  
grant SELECT, REPLICATION CLIENT, REPLICATION SLAVE, PROCESS on *.* to 'migration acc  
flush privileges;
```

## Restrictions

Currently, the subscribed message content is retained for 1 day by default. Once expired, the data will be cleared. Therefore, you need to consume the data promptly.

The region where the data is consumed should be the same as that of the subscribed instance.

Geometry data types are not supported currently.

If the data subscription source is TDSQL for MySQL, it is not supported to directly run the authorization statement.

Instead, you need to go to the [TDSQL console](#), click the target instance ID, and enter the account management page to authorize.

The permissions required by the subscription account are those listed in the above authorization statement. To perform `__tencentdb__` authorization to the subscription account, select **Object-Level Privilege** in the **Modify Permissions** pop-up window and then select all permissions.

If the data subscription source is TDSQL for MySQL, [two-level partitioned tables](#) cannot be subscribed to.

If a two-level partitioned table is created in the source database before the subscription task is started, the verification task will fail.

If a two-level partitioned table is created in the source database when the subscription task is running, the data subscribed to in the two-level partitioned table will be the data in the child table (if the selected subscription object is the entire database or entire instance, a two-level partitioned table created in the source database after the subscription task is started will also be subscribed to; in this way, data in the two-level partitioned table will also be included in the subscription). As the underlying layer of the two-level partitioned table is implemented by child tables, we recommend that you not create two-level partitioned tables during the subscription task execution; otherwise, differences in the subscribed data as shown below will occur.

For example, if the source database table "test\_a" is a two-level partitioned table, then the table name of the DML that DTS subscribes to this table is "test\_a\_tdsql\_subp0/test\_a\_tdsql\_subp1".

During the process of subscribing to a task, if you perform operations such as modifying the subscription object, the task will be restarted, which may cause duplicated data consumption on the Kafka client.

DTS transfers data based on the smallest data unit. Every time the incremental data is marked with a checkpoint, it will be considered a data unit. If the transfer of one data unit is completed before the task restart, it will not cause data duplication; if one data unit is still being transferred during the task restart, the data unit needs to be pulled again after the restart to ensure data integrity, which will lead to data duplication.

If you are more concerned about duplicate data, you can implement deduplication logic when consuming data.

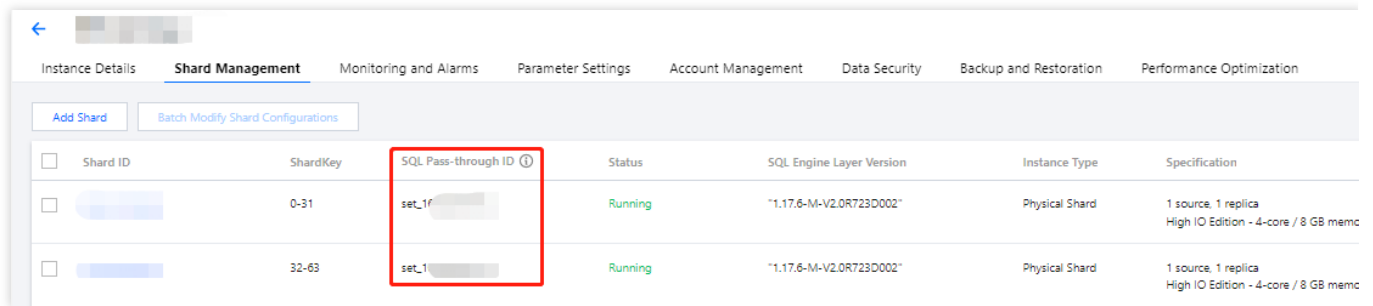
## Note

For a subscription task whose database for subscription is TDSQL for MySQL, the DDL operations of each shard will be subscribed to and be published to Kafka. Therefore, DDL operations of a sharded table will lead to duplicate DDL statements. For example, if instance A has 3 shards and you subscribe to a sharded table A, you will subscribe to 3 DDL statements of table A.

The header of each message in Kafka carries the shard information in `key/value` form. Here, the key is

`ShardId`, and the value is the SQL passthrough ID. You can identify the shard where the message comes from

based on the SQL passthrough ID. You can check the SQL passthrough ID in the [TDSQL console](#): click **Instance List > Manage** in the **Operation** column, and select the **Shard Management** tab to view \*SQL Passthrough ID\*\*.



Instance Details <b>Shard Management</b> Monitoring and Alarms Parameter Settings Account Management Data Security Backup and Restoration Performance Optimization							
Add Shard Batch Modify Shard Configurations							
<input type="checkbox"/>	Shard ID	ShardKey	SQL Pass-through ID ⓘ	Status	SQL Engine Layer Version	Instance Type	Specification
<input type="checkbox"/>		0-31	set_1f	Running	"1.17.6-M-V2.0R723D002"	Physical Shard	1 source, 1 replica High IO Edition - 4-core / 8 GB memc
<input type="checkbox"/>		32-63	set_1	Running	"1.17.6-M-V2.0R723D002"	Physical Shard	1 source, 1 replica High IO Edition - 4-core / 8 GB memc

## Subscribable SQL Operations

Operation Type	Supported SQL Operations
DML	INSERT, UPDATE, DELETE
DDL	CREATE DATABASE、DROP DATABASE、CREATE TABLE、ALTER TABLE、DROP TABLE、RENAME TABLE

## Directions

1. Log in to the [DTS console](#), select **Data Subscription** on the left sidebar, and click **Create Subscription**.

2. On the **Create Subscription** page, select the corresponding configuration and click **Buy Now**.

Billing Mode: Monthly subscription and pay-as-you-go billing are supported.

Region: The region must be the same as that of the database instance to be subscribed to.

Database: Select your actual database type.

Version: Select **Kafka Edition**. You can directly consume data on a Kafka client.

Subscribed Instance Name: Edit the name of the current data subscription instance.

3. After successful purchase, return to the data subscription list. You need to click **Configure Subscription** in the **Operation** column to configure the newly purchased subscription before you can use it.

4. On the **Subscription Configuration** page, select the appropriate configuration items and click **Next**.

Database Instance: Select a database instance. You can select a disaster recovery read-only instance. However, it is recommended to select the source instance, so that the pressure of the subscription service on the source database is minimized.

Database Account: add the account and password of the instance to be subscribed to. The account must have the permissions required by the subscription task, including REPLICATION CLIENT, REPLICATION SLAVE, PROCESS, and SELECT of all objects.

Number of Kafka Partitions: Set the number of Kafka partitions. Increasing the number can improve the speed of data write and consumption. A single partition can guarantee the order of messages, while multiple partitions cannot. If you have strict requirements for the order of messages during consumption, set this value to 1.

5. On the **Subscription Type and Object** page, select a subscription type and click **Save**.

Subscription Type: Options include **Data Update**, **Structure Update**, and **Full**.

Data Update: data updates of the selected objects are subscribed to, including INSERT, UPDATE, and DELETE operations.

Structure Update: creation, modification, and deletion of the structures of all objects in the instance are subscribed to.

Full: data and structure updates of all objects in the instance are subscribed to.

6. On the **Pre-verification** page, a pre-verification task will run for 2–3 minutes. After the pre-verification is passed, click **Start** to complete data subscription task configuration.

#### Note

If the verification fails, fix the problem as instructed in [Check Item Overview](#) and initiate the verification again.

7. After you click **Start**, the subscription task will be initialized, which will take 3–4 minutes. After successful initialization, the task will enter the **Running** status.

8. Add a consumer group. Data subscription (Kafka Edition) allows you to create multiple consumer groups for multi-point consumption. For more information, see [Adding Consumer Group](#). The consumption in data subscription (Kafka Edition) depends on the consumer groups of Kafka; therefore, you must create a consumer group first before data can be consumed.

9. After the subscription instance enters the **Running** status, you can start consuming data. For consumption in Kafka, you need to verify the password. For specific examples, see Data Consumption Demo. We provide demo code in multiple programming languages and descriptions of main consumption processes and key data structures.



# Creating MariaDB/Percona Data Subscription

Last updated : 2024-07-08 16:52:02

This document describes how to create a data subscription task in DTS for MariaDB and Percona. MariaDB is used as an example, and the related operations also apply to Percona.

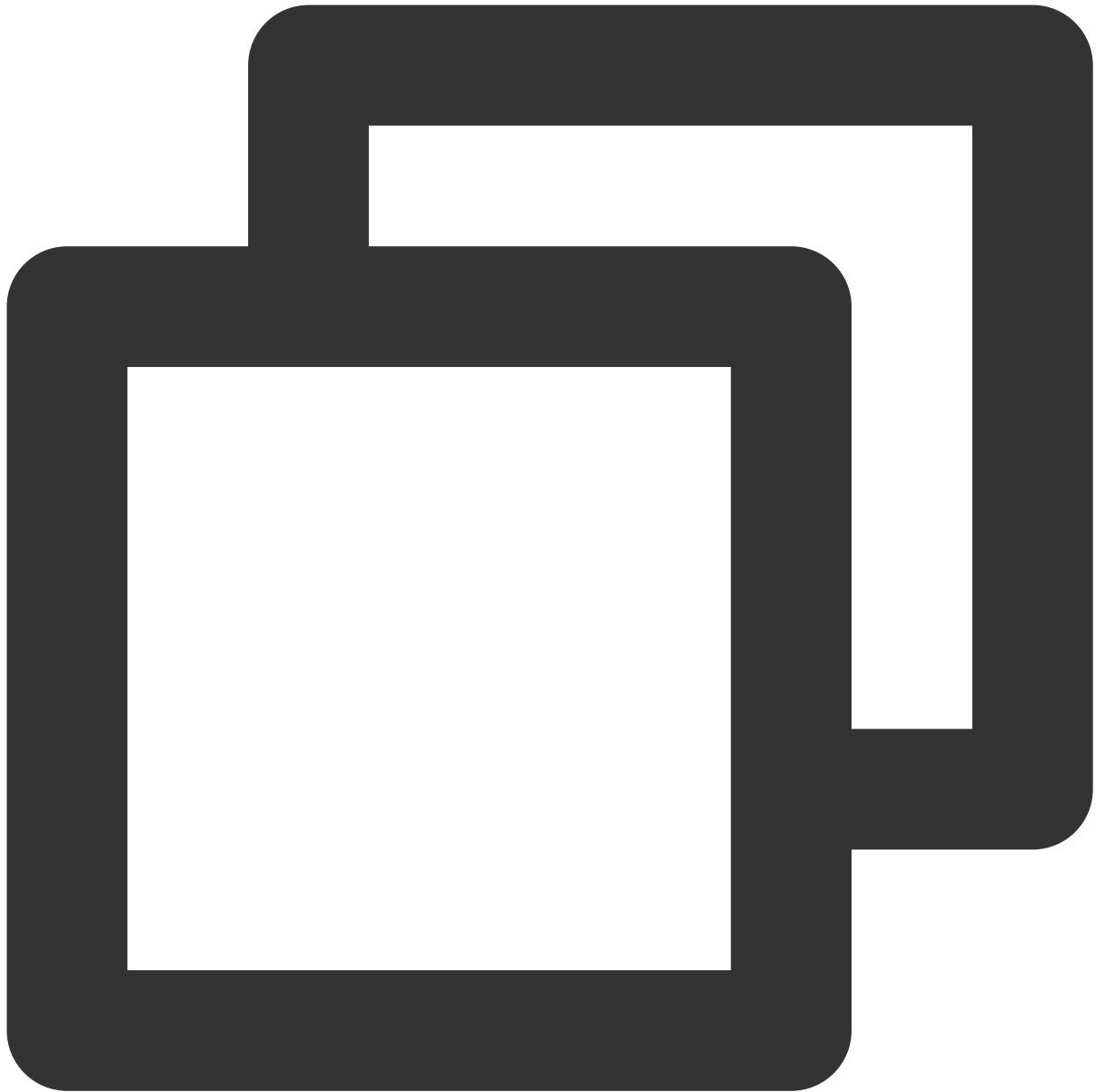
## Prerequisites

You have prepared a source database to be subscribed to, and the database version meets the requirements. For more information, see [Databases Supported by Data Subscription](#).

You have enabled the binlog in the source database.

You have created a subscription account in the source database and granted the account the following permissions: REPLICATION CLIENT, REPLICATION SLAVE, PROCESS, and SELECT for all objects.

Authorization statements are as follows:



```
CREATE USER 'account' IDENTIFIED BY 'password';  
GRANT SELECT, REPLICATION CLIENT, REPLICATION SLAVE, PROCESS ON *.* TO 'account'@'%';  
FLUSH PRIVILEGES;
```

## Restrictions

Currently, the subscribed message content is retained for one day by default. The data will be cleared when the retention period ends. Therefore, you need to consume the data promptly.

The region where the data is consumed should be the same as that of the subscribed instance.

Geometry data types are not supported currently.

During the process of subscribing to a task, if you perform operations such as modifying the subscription object, the task will be restarted, which may cause duplicated data consumption on the Kafka client.

DTS transfers data based on the smallest data unit. Every time the incremental data is marked with a checkpoint, it will be considered a data unit. If the transfer of one data unit is completed before the task restart, it will not cause data duplication; if one data unit is still being transferred during the task restart, the data unit needs to be pulled again after the restart to ensure data integrity, which will lead to data duplication.

If you are more concerned about duplicate data, you can implement deduplication logic when consuming data.

## Subscribable SQL Operations

Operation Type	Supported SQL Operations
DML	INSERT, UPDATE, DELETE
DDL	CREATE DATABASE, DROP DATABASE, CREATE TABLE, ALTER TABLE, DROP TABLE, RENAME TABLE

## Directions

1. Log in to the [DTS console](#), select **Data Subscription** on the left sidebar, and click **Create Subscription**.

2. On the **Create Subscription** page, select the corresponding configuration and click **Buy Now**.

Billing Mode: Monthly subscription and pay-as-you-go billing are supported.

Region: If the source database is a TencentDB database, select the region of the source database instance here.

Otherwise, select the region where data is consumed. If there is no special requirement for the data consumption region, select a region closest to the source database.

Database: Select the database type.

Edition: Select **Kafka Edition**. You can directly consume data on a Kafka client.

Subscribed Instance Name: Edit the name of the currently subscribed instance.

3. After successful purchase, return to the data subscription list. You need to click **Configure Subscription** in the **Operation** column to configure the newly purchased subscription before you can use it.

4. On the subscription configuration page, select the source database information, click **Test Connectivity**, and click **Next** after the test is passed.

Configuration Item	Description

Instance Type	<p>The database type selected during purchase.</p> <p>We recommend that you subscribe to the primary database because the subscription service only puts a little pressure on the source database.</p> <p>If the source database is TencentDB for MariaDB, you can select a disaster recovery or read-only instance.</p>
Access Type	<p>Select a type based on your scenario. For the preparations for different access types, see <a href="#">Preparations &gt; Overview</a>.</p> <p>Public Network: The source database can be accessed through a public IP.</p> <p>Self-Build on CVM: The source database is deployed on a CVM instance. For more information on CVM, see <a href="#">Cloud Virtual Machine (CVM)</a>.</p> <p>Direct Connect: The source database can be interconnected with VPCs through Direct Connect. For more information on Direct Connect, see <a href="#">Direct Connect</a>.</p> <p>VPN Access: The source database can be interconnected with VPCs through VPN Connections. For more information on VPN Connections, see <a href="#">VPN Connections</a>.</p> <p>Database: The source database is a TencentDB database.</p> <p>CCN: The source database can be interconnected with VPCs through CCN. For more information on CCN, see <a href="#">Cloud Connect Network (CCN)</a>.</p>
Public Network	<p>Host Address: IP address or domain name of the source database.</p> <p>Port: Port used by the source database.</p>
Self-Build on CVM	<p>CVM Instance: The ID of the CVM instance.</p> <p>Port: Port used by the source database.</p>
Direct Connect	<p>VPC-Based Direct Connect Gateway: Only VPC-based direct connect gateway is supported. Confirm the network type associated with the gateway.</p> <p>VPC: Select a VPC and subnet associated with the VPC-based Direct Connect gateway or VPN gateway.</p> <p>Host Address: IP address of the source database.</p> <p>Port: Port used by the source database.</p>
VPN Access	<p>VPN Gateway: Select a VPN gateway instance.</p> <p>VPC: Select a VPC and subnet associated with the VPC-based Direct Connect gateway or VPN gateway.</p> <p>Host Address: IP address of the source database.</p> <p>Port: Port used by the source database.</p>
Database	<p>Instance Name: The ID of the source database instance.</p>
CCN	<p>Host Address: IP address of the source database server.</p> <p>Port: Port used by the source database.</p> <p>VPC-Based CCN Instance: The name of the CCN instance.</p> <p>Accessed VPC: It refers to the VPC in CCN over which the subscription link is connected. You need to select a VPC other than the VPC to which the source database belongs.</p>

	<p>For example, if the database in Guangzhou is used as the source database, select another region, such VPC-Chengdu or VPC-Shanghai, as the accessed VPC.</p> <p>Subnet: Name of the subnet of the selected VPC.</p> <p>Region of Accessed VPC: The region of the source database selected during task purchase must be the same as the region of the accessed VPC; otherwise, DTS will change the former to the latter.</p>
Account/Password	Account/Password: Enter the database account and password.
Number of Kafka Partitions	Set the number of Kafka partitions. Increasing the number can improve the speed of data write and consumption. A single partition can guarantee the order of messages, while multiple partitions cannot. If you have strict requirements for the order of messages during consumption, set this value to 1.

5. On the **Subscription Type and Object** page, select a subscription type and click **Save**.

Subscription Type: Options include **Data Update**, **Structure Update**, and **Full Instance**.

Data Update: Data updates of the selected objects are subscribed to, including INSERT, UPDATE, and DELETE operations.

Structure Update: creation, modification, and deletion of the structures of all objects in the instance are subscribed to.

Full Instance: Data and structure updates of all objects in the instance are subscribed to.

Kafka Partitioning Policy: Select **By table name** or **By table name + primary key**.

6. On the **Pre-verification** page, a pre-verification task will run for 2–3 minutes. After the pre-verification is passed, click **Start** to complete the data subscription task configuration.

#### Note

If the verification fails, fix the problem as instructed in [Check Item Overview](#) and initiate the verification again.

7. The subscription task will be initialized, which will take 3–4 minutes. After successful initialization, the task will enter the **Running** status.

8. Add a consumer group. Data subscription (Kafka Edition) allows you to create multiple consumer groups for multi-point consumption. For more information, see [Creating Consumer Group](#). The consumption in data subscription (Kafka Edition) depends on the consumer groups of Kafka; therefore, you must create a consumer group first before data can be consumed.

9. After the subscription instance enters the **Running** status, you can start consuming data. To consume data in Kafka, you need to verify the password. For specific examples, see [Consuming MySQL Data](#). We provide demo code in multiple programming languages and descriptions of main consumption processes and key data structures.

# Consuming MySQL Data

## Operation Guide for Consuming MySQL Series Subscribed Data

Last updated : 2024-07-08 16:52:02

### Overview

In data subscription (Kafka Edition, where the current Kafka Server version is 2.6.0), you can consume the subscribed data through Kafka 0.11 or later available at [DOWNLOAD](#). This document provides client consumption demos for Java, Go, and Python for you to quickly test the process of data consumption and understand the method of data parsing.

When configuring the subscription task, you can select different formats of subscribed data, including ProtoBuf, Avro, and JSON. ProtoBuf and Avro adopt the binary format with a higher consumption efficiency, while JSON adopts the easier-to-use lightweight text format. The reference demo varies by the selected data format.

This document provides a demo of the Avro format. The demo already contains the Avro protocol file, so you don't need to download it separately.

#### Notes

Currently, data consumption over the Avro protocol is supported only for TencentDB for MySQL and TDSQL-C for MySQL.

### Notes

**The demo only prints out the consumed data and does not contain any usage instructions. You need to write your own data processing logic based on the demo.** You can also use Kafka clients in other languages to consume and parse data.

Currently, data subscription to Kafka for consumption can be implemented over the Tencent Cloud private network but not the public network. In addition, the subscribed database instance and the data consumer must be in the same region.

The Kafka built in DTS has a certain upper limit for processing individual messages. When a single row of data in the source database exceeds 10 MB, this row may be discarded.

### Downloading a consumption demo

Demo Language	ProtoBuf (MySQL/MariaDB/TDSQL-C MySQL/Percona)	ProtoBuf (TDSQL MySQL)	Avro (MySQL/TDSQL-C MySQL)	JSON (MySQL/TDSQL-C MySQL)
Go	<a href="#">Address</a>	<a href="#">Address</a>	<a href="#">Address</a>	<a href="#">Address</a>
Java	<a href="#">Address</a>	<a href="#">Address</a>	<a href="#">Address</a>	<a href="#">Address</a>
Python	<a href="#">Address</a>	<a href="#">Address</a>	<a href="#">Address</a>	<a href="#">Address</a>

## Instructions for the Java demo

Compilation environment: Maven or Gradle and JDK 8. You can choose a desired package management tool. The following takes Maven as an example.

Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install JRE 8.

Directions:

1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).
2. Create one or multiple consumer groups. For more information, see [Adding Consumer Group](#).
3. Download the Java demo and decompress it.
4. Access the decompressed directory. Maven model and pom.xml files are placed in the directory for your use as needed.

Package with Maven by running `mvn clean package`.

5. Run the demo.

After packaging the project with Maven, go to the target folder `target` and run `java -jar consumerDemo-avro-1.0-SNAPSHOT.jar --brokers xxx --topic xxx --group xxx --user xxx --password xxx --trans2sql`.

`broker` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement. In Java code, if this parameter is carried, the conversion will be enabled.

### Notes

If `trans2sql` is carried, `javax.xml.bind.DatatypeConverter.printHexBinary()` will be used to convert byte values to hex values. You should use JDK 1.8 or later to avoid incompatibility. If you don't need SQL conversion, comment this parameter out.

## 6. Observe consumption.

```
BEGIN
-->[partition: 0, offset: 87008, partitionSeq: 87009] [mysql-bin.000004:24574], happenedAt: 2021-03-01T
17:49:14
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 61)
-->[partition: 0, offset: 87008, partitionSeq: 87009] [mysql-bin.000004:24605], happenedAt: 2021-03-01T
17:49:14
COMMIT
```

## Instructions for the Go demo

Compiling environment: Go 1.12 or later, with the Go module environment configured.

Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance).

Directions:

1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).

2. Create one or multiple consumer groups. For more information, see [Adding Consumer Group](#).

3. Download the Go demo and decompress it.

4. Access the decompressed directory and run `go build -o subscribe ./main/main.go` to generate the executable file `subscribe`.

5. Run `./subscribe --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=true`.

`broker` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement.

## 6. Observe consumption.

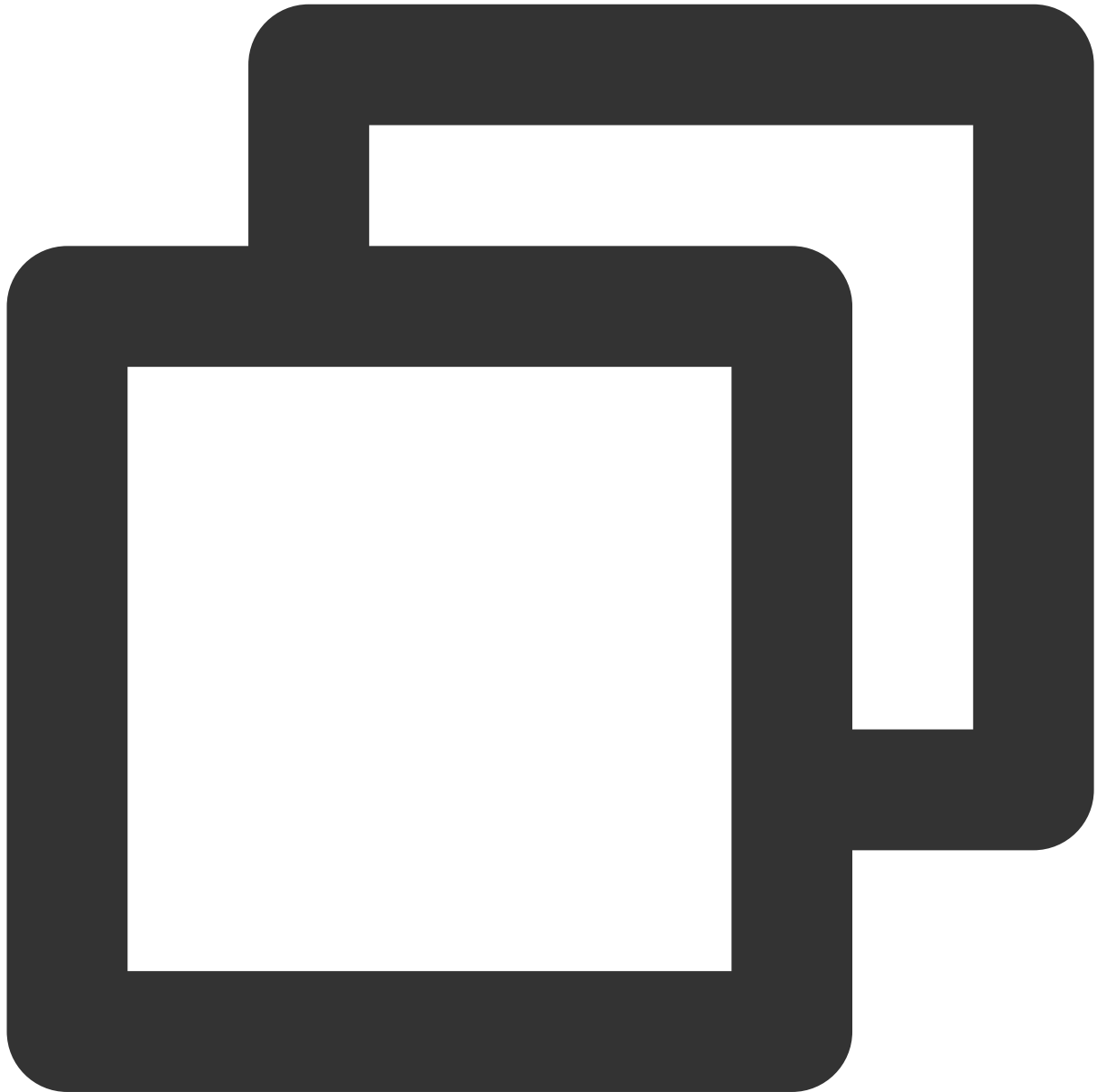
```
BEGIN
-->[partition: 0, offset: 86991, partitionSeq: 86992] [mysql-bin.000004:24272], happenedAt: 2021-03-01
17:47:49 +0800 CST
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 60)
-->[partition: 0, offset: 86991, partitionSeq: 86992] [mysql-bin.000004:24303], happenedAt: 2021-03-01
17:47:49 +0800 CST
COMMIT
```

## Instructions for the Python 3 demo



Compilation and runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install Python 3 and pip3 (for dependency package installation).

Use `pip3` to install the dependency package:



```
pip install flag
pip install kafka-python
pip install avro
```

Directions:

1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).

2. Create one or multiple consumer groups. For more information, see [Adding Consumer Group](#).

3. Download the Python 3 demo and decompress it.

4. Run `python main.py --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=1`.

`broker` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement.

5. Observe consumption.

```
BEGIN
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24876], happenedAt: 2021-03-01
20:43:31
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 62)
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24907], happenedAt: 2021-03-01
20:43:31
COMMIT
```

# Demo Description

## ProtoBuf Demo Description

Last updated : 2024-07-08 16:52:02

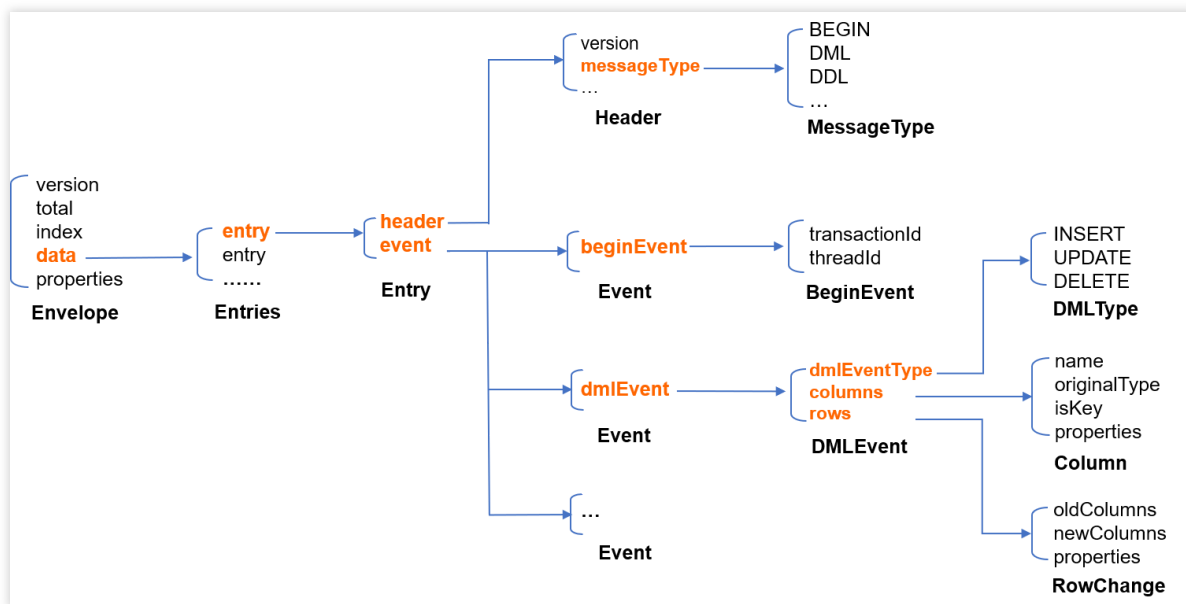
## Key Logic Description

### Message production logic

This section describes the message production logic to help you better understand the consumption logic.

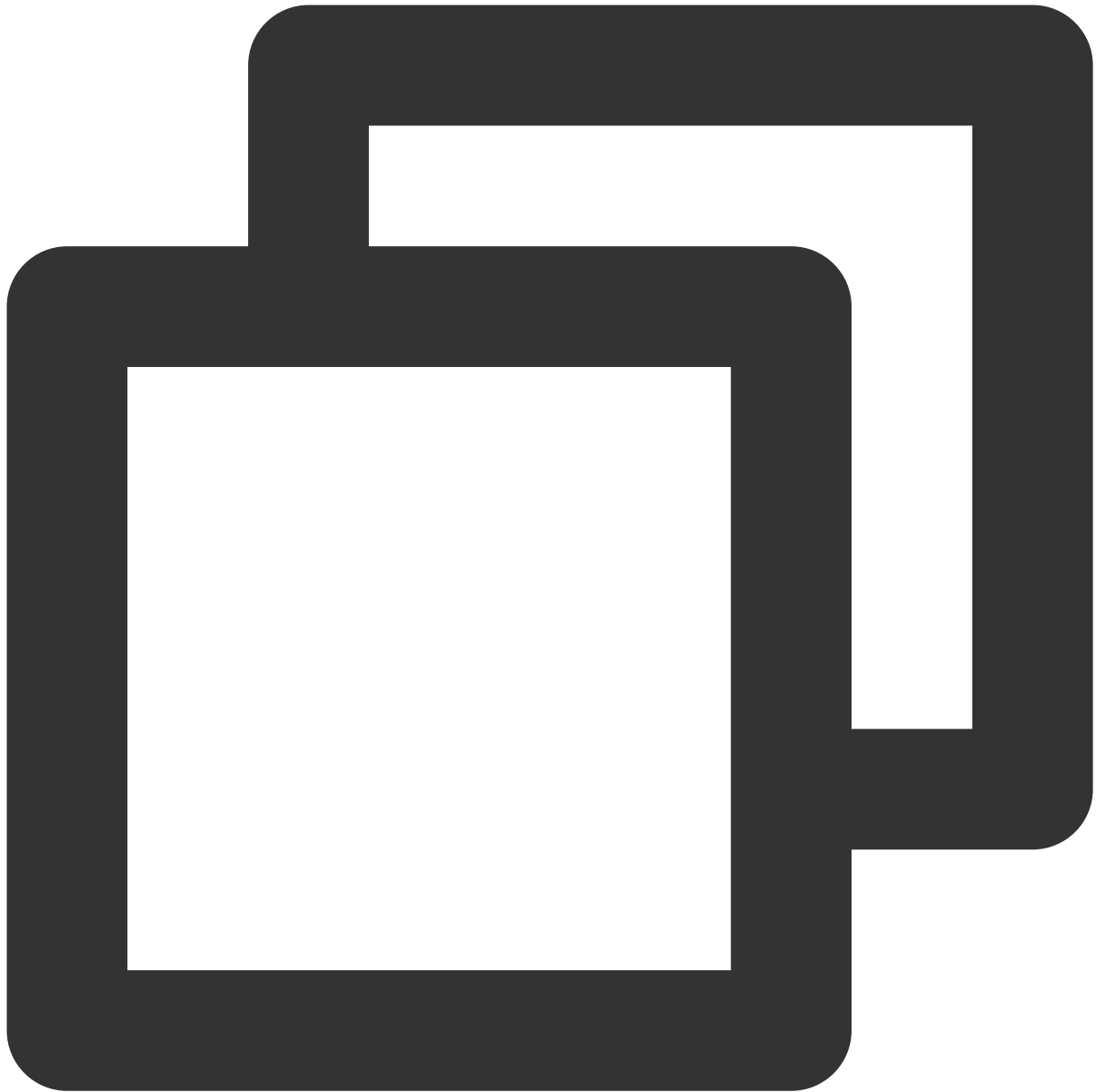
The demo for each programming language uses Protobuf for serialization and contains a `Protobuf` definition file.

In the file, three key structures are defined as follows: `Envelope` is the final Kafka message structure; `Entry` is the structure of a single subscription event; `Entries` is the collection of `Entry`. Their relationship is shown below:



The production process is as follows:

1. Pull binlog messages and encode each binlog event into an `Entry`.



```
message Entry { // An `Entry` is the structure of an individual subscription event.
  Header header = 1; // The event header
  Event event   = 2; // The event body
}
```

```
message Header {
  int32      version      = 1; // The protocol version of the `Entry`
  SourceType sourceType   = 2; // The source database type, such as MySQL and O
  MessageType messageType = 3; // The message type, i.e., event type, such as B
  uint32 timestamp        = 4; // The event timestamp in the source binlog
}
```

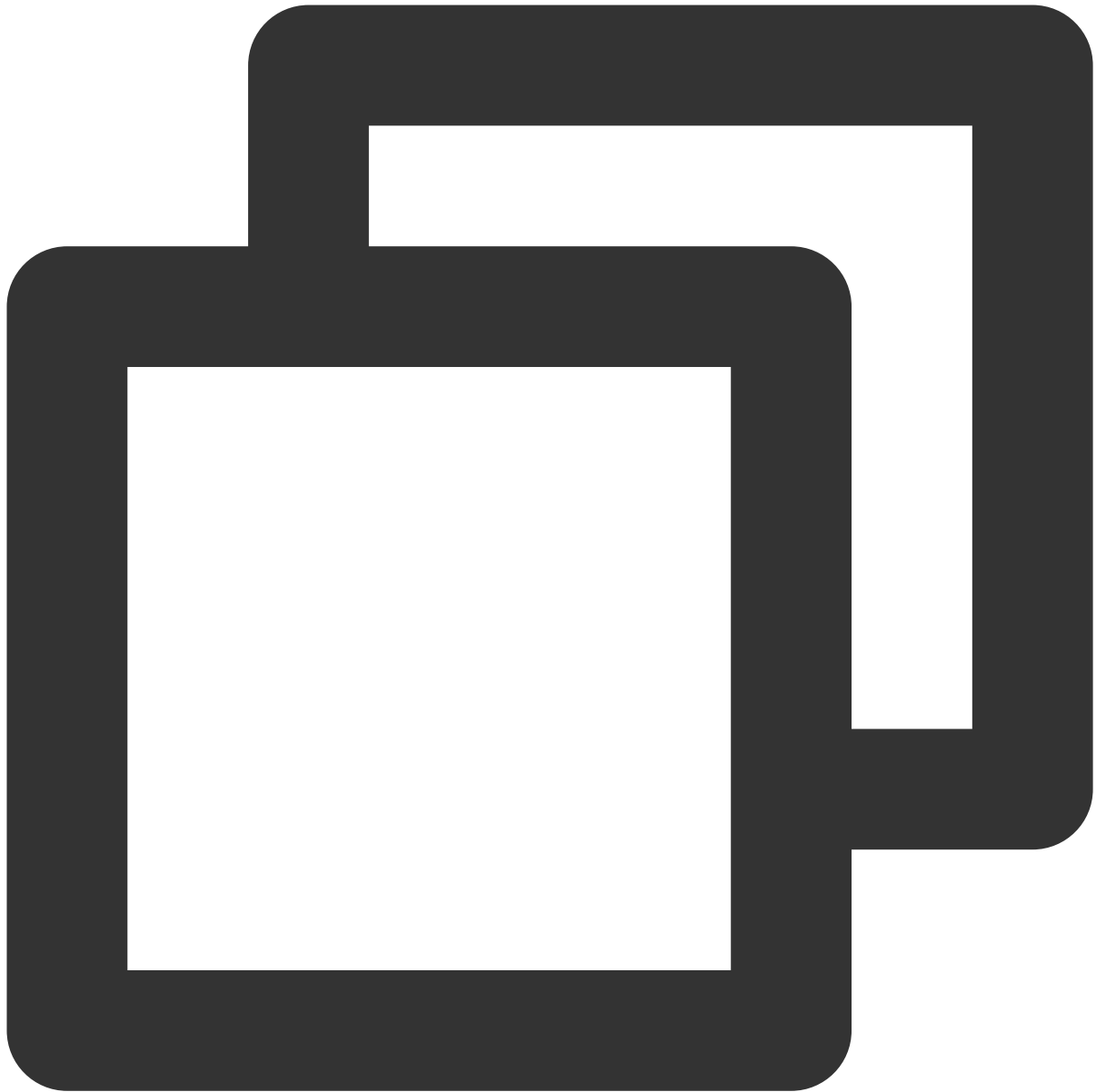
```

int64  serverId      = 5;    // The `serverId` of the source database
string fileName      = 6;    // The filename of the source binlog
uint64 position      = 7;    // The event offset in the source binlog file
string gtId          = 8;    // The GTID of the current transaction
string schemaName    = 9;    // The modified schema
string tableName     = 10;   // The modified table
uint64 seqId         = 11;   // The globally incremental serial number
uint64 eventIndex    = 12;   // If a large event is sharded, the shard number
bool   isLast        = 13;   // Whether the current shard is the last shard o
repeated KVPair properties = 15;
}

message Event {
  BeginEvent      beginEvent      = 1;  // The BEGIN event in the binlog
  DMLEvent        dmlEvent        = 2;  // The DML event in the binlog
  CommitEvent     commitEvent     = 3;  // The COMMIT event in the binlog
  DDLEvent        ddlEvent        = 4;  // The DDL event in the binlog
  RollbackEvent   rollbackEvent   = 5;  // The rollback event. This parameter is mea
  HeartbeatEvent  heartbeatEvent  = 6;  // The heartbeat event regularly sent by the
  CheckpointEvent checkpointEvent = 7;  // The checkpoint event added to the subscri
  repeated KVPair properties      = 15;
}

```

2. Multiple `Entry` structures are merged to reduce the number of messages, and the structure of binlog events becomes `Entries` after the merge. The `Entries.items` field refers to the `Entry` sequence list. The reasonable number of merged `Entry` structures should be smaller than that of a single Kafka message. If a single binlog event has exceeded the size limit, `Entry` structures will not be merged anymore, so there will be only one `Entry` in the `Entries` structure.

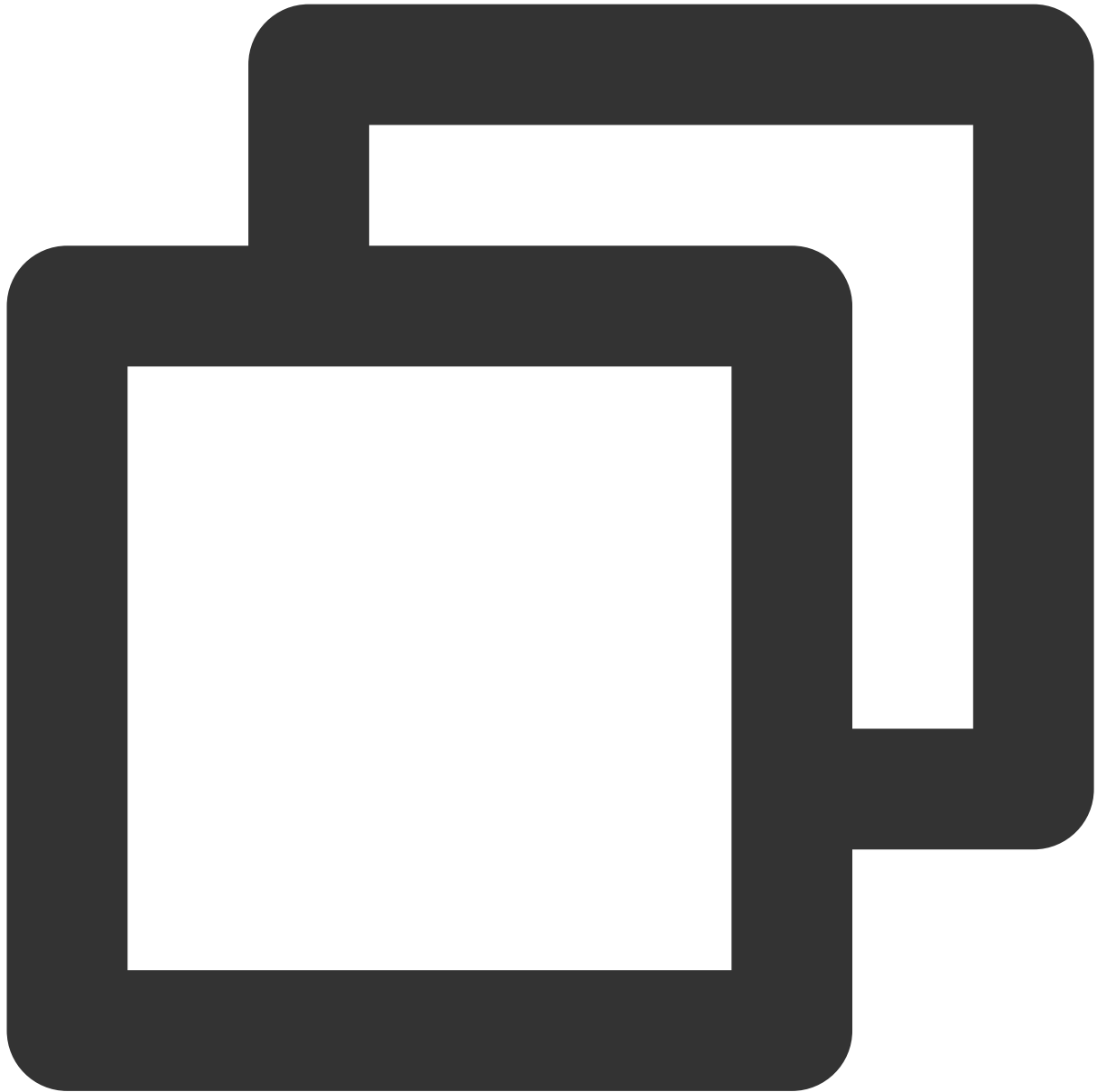


```
message Entries {  
    repeated Entry items = 1; // `Entry` list  
}
```

3. Encode `Entries` with Protobuf to generate a binary sequence.

4. Put the binary sequence in the `data` field of an `Envelope`. If a single binlog event is oversize, the binary sequence may exceed the size limit of a single Kafka message. In this case, you can separate the binary sequence into multiple segments and put each segment in an `Envelope`.

`Envelope.total` and `Envelope.index` record the total number of `Envelope` structures and the serial number of the current `Envelope` structure (starting from 0) respectively.



```
message Envelope {  
    int32  version           = 1; // The protocol version, which determines  
    uint32 total             = 2;  
    uint32 index             = 3;  
    bytes  data              = 4; // Here, `version` is 1, indicating that  
    repeated KVPair properties = 15;  
}
```

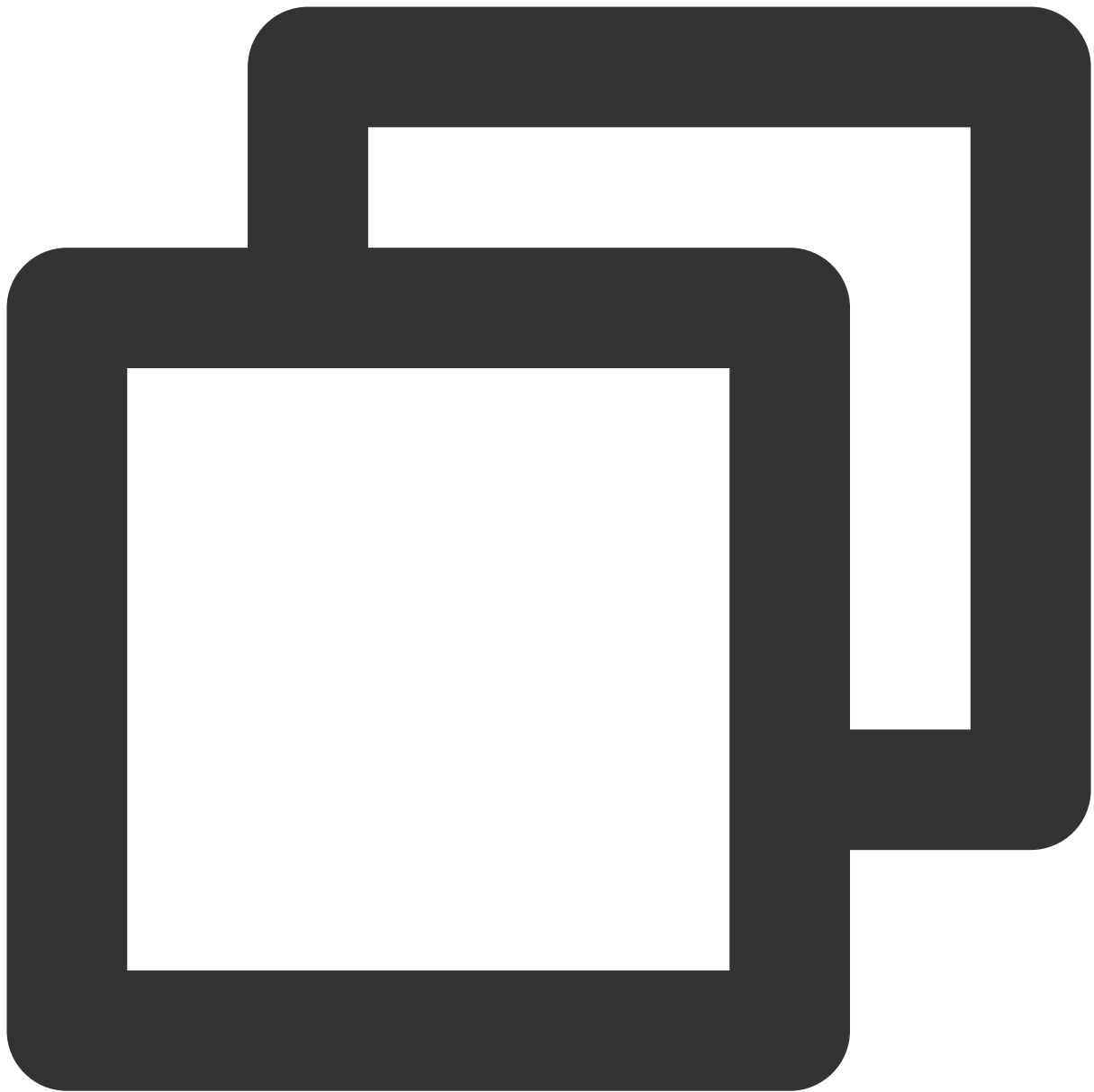
5. Encode one or multiple `Envelope` structures generated in the previous step in sequence and deliver the `Envelope` structures to Kafka partitions. Multiple `Envelope` structures in the same `Entries` are delivered to the same partition in sequence.

## Message consumption logic

This section describes the message consumption logic. The description here applies to our demos for Java, Go, and Python.

1. Create a Kafka consumer.
2. Start the consumption.
3. Consume original messages in sequence and find the corresponding `partitionMsgConsumer` object of a message partition to process these messages.
4. The `partitionMsgConsumer` object deserializes messages into the `Envelope` structure.

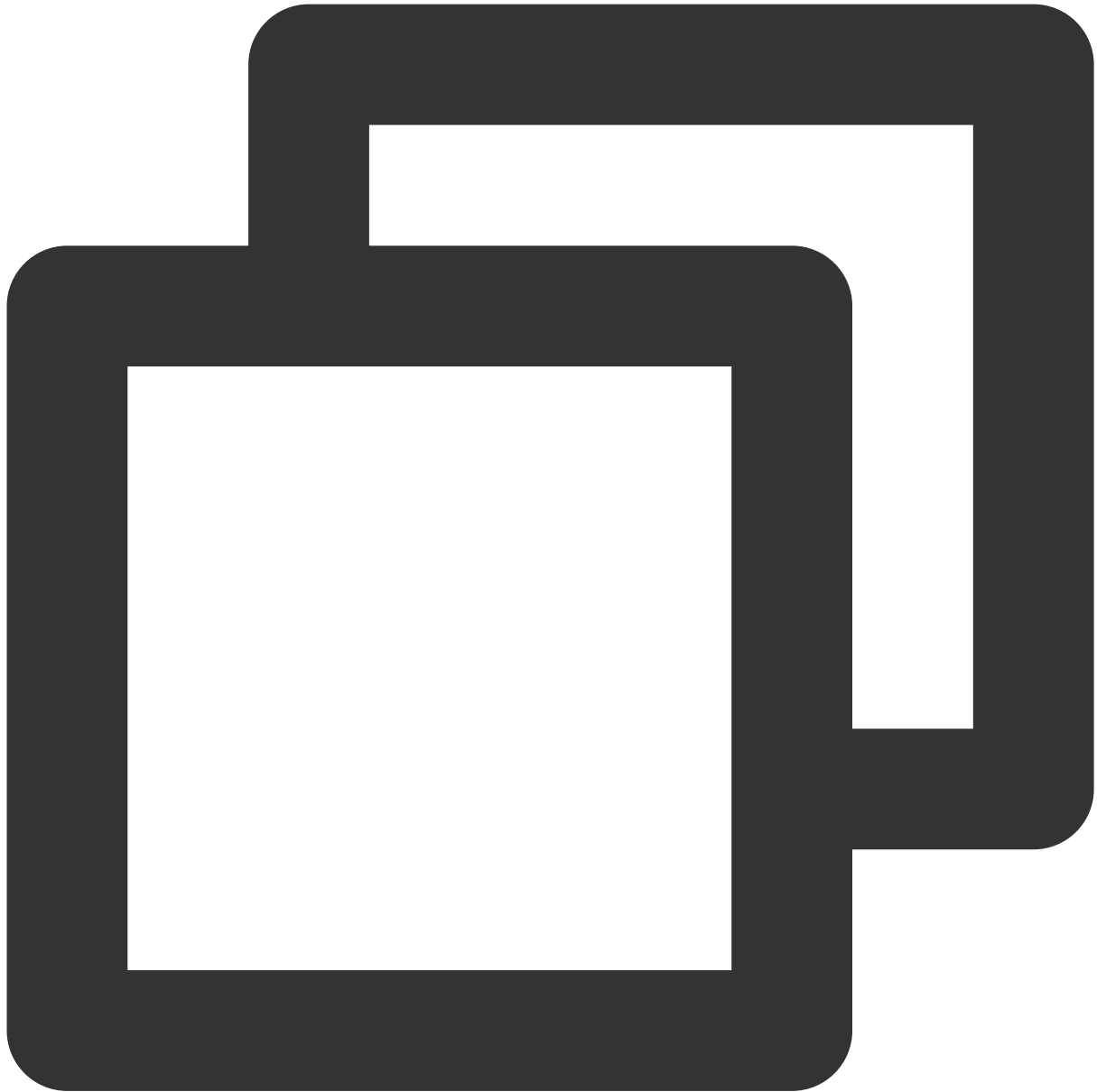




```
// Convert the value of the Kafka message to an `Envelope`.
envelope := subscribe.Envelope{}
err := proto.Unmarshal(msg.Value, &envelope)
```

5. The `partitionMsgConsumer` object continuously consumes one or multiple messages based on the `index` and `total` recorded in the `Envelope` until `Envelope.index` equals to `Envelope.total-1` (which indicates that a complete `Entries` is received. See the consumption and production logic mentioned above).

6. Combine the `data` fields of multiple consecutive `Envelope` structures received together in sequence. Decode the combined binary sequences into `Entries` with Protobuf.



```
if envelope.Index == 0 {
    pmc.completeMsg = envelope
} else {
    // Splice the split binary sequences of `Entries`
    pmc.completeMsg.Data = append(pmc.completeMsg.Data, envelope.Data...)
}
if envelope.Index < envelope.Total-1 {
    return nil
}
```

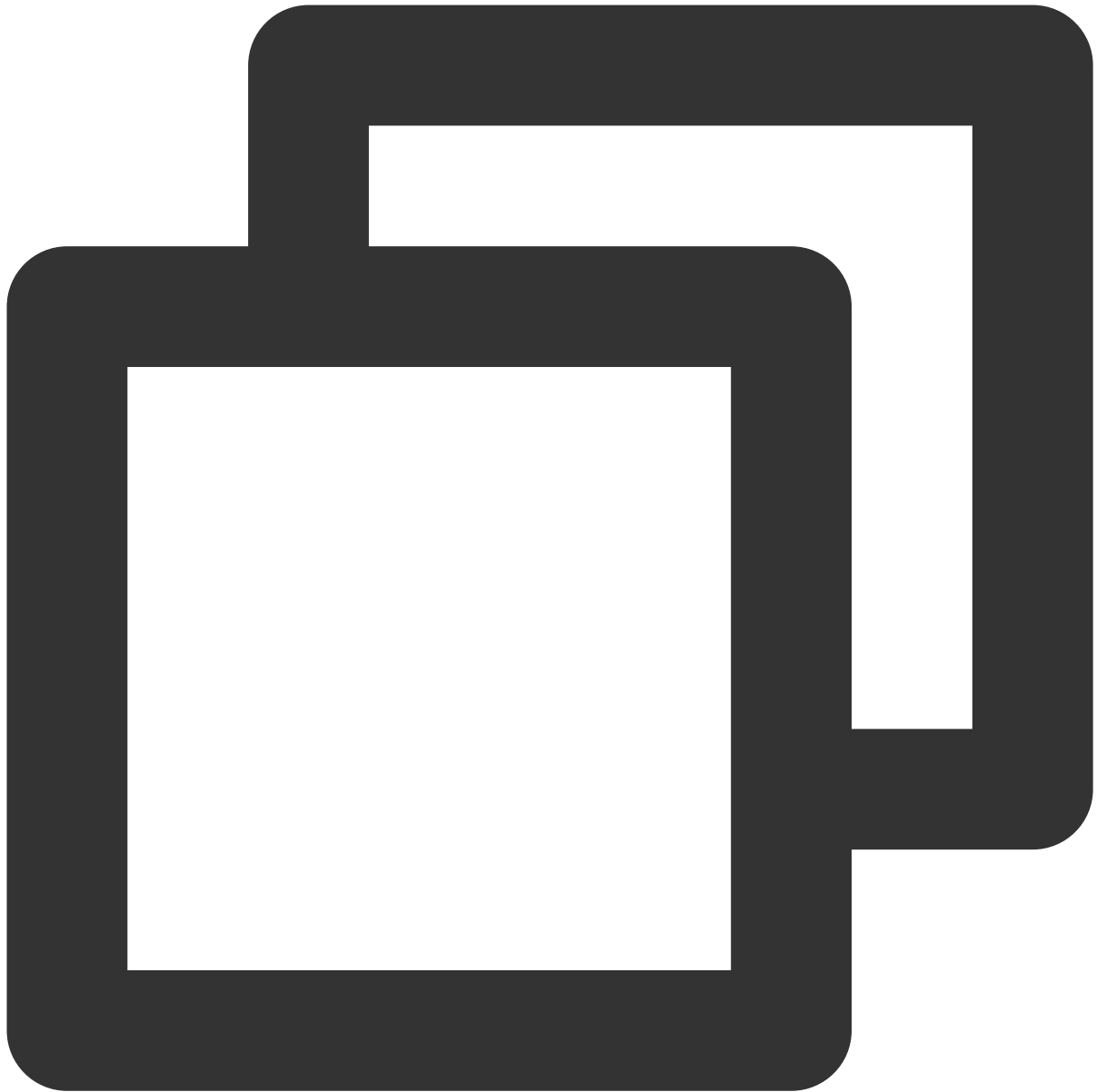
```
}  
// Deserialize `Envelope.Data` to `Entries`  
entries := subscribe.Entries{}  
err = proto.Unmarshal(pmc.completeMsg.Data, &entries)
```

7. Process `Entries.items` in sequence, and print the original `Entry` structure or convert it into a SQL statement.

8. When a checkpoint message is consumed, submit the Kafka message offset. A checkpoint message is a special message written to Kafka by the subscription backend once every 10 seconds.

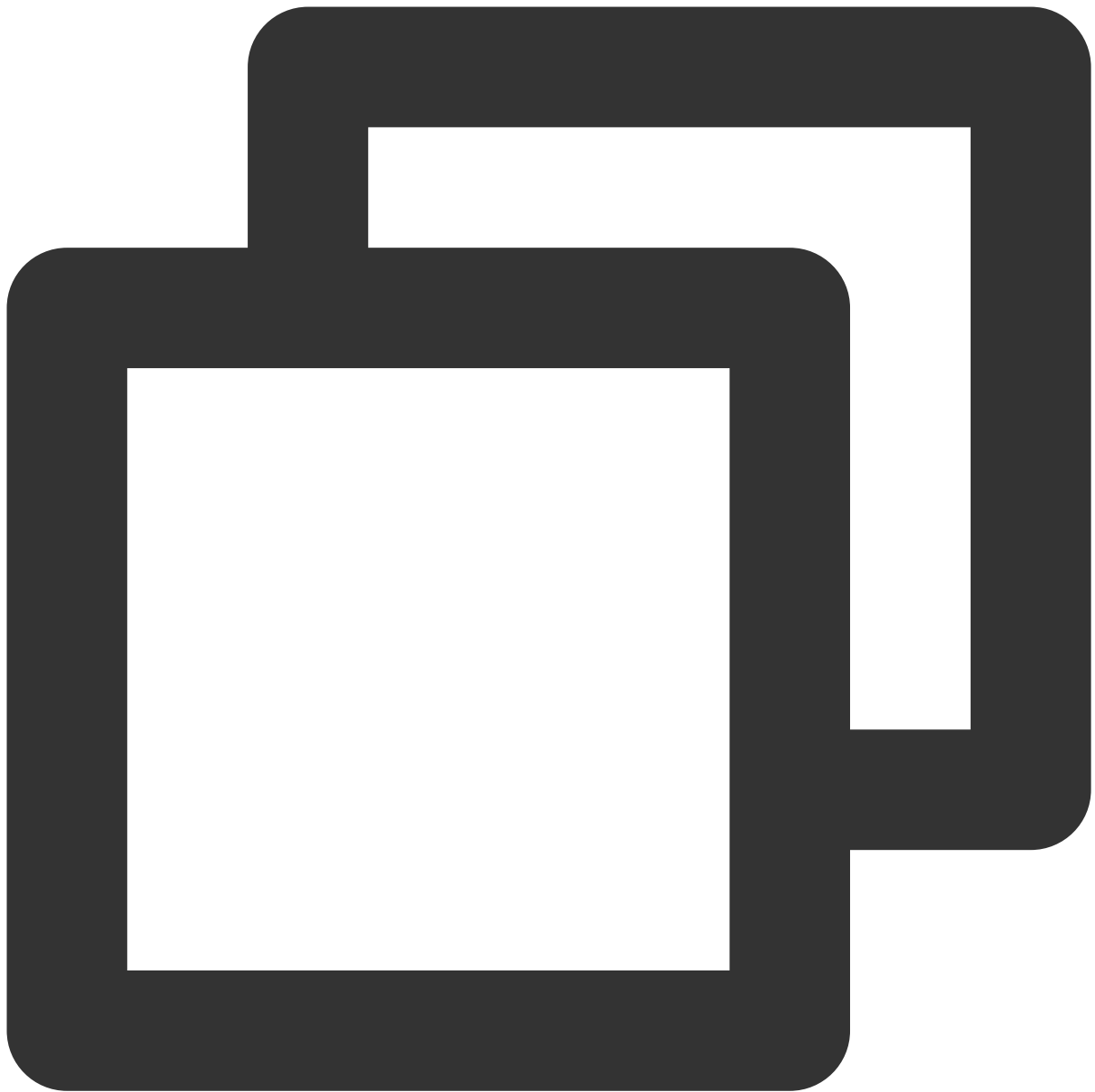
## Database Field Mapping and Storage

This section describes the mappings between database field types and data types defined in the Protobuf protocol. A field value in the source database such as MySQL is structured as follows in the Protobuf protocol.



```
message Data {  
    DataType      dataType = 1;  
    string        charset  = 2; // The encoding (string) type of DataType_STRING,  
    string        sv       = 3; // The string value of DataType_INT8/16/32/64/UINT  
    bytes         bv       = 4; // The value of DataType_STRING/DataType_BYTES  
}
```

The field `DataType` refers to the type of stored fields. The values are as enumerated below:



```
enum DataType {  
    NIL      = 0; // The value is `NULL`  
    INT8     = 1;  
    INT16    = 2;  
    INT32    = 3;  
    INT64    = 4;  
    UINT8    = 5;  
    UINT16   = 6;  
    UINT32   = 7;  
    UINT64   = 8;  
    FLOAT32  = 9;
```

```

    FLOAT64 = 10;
    BYTES   = 11;
    DECIMAL = 12;
    STRING  = 13;
    NA      = 14; // The value does not exist (N/A).
}

```

The `bv` field stores the binary representation of `STRING` and `BYTES`; the `sv` field stores the string representation of `INT8/16/32/64/UINT8/16/32/64/DECIMAL`; the `charset` field stores the encoding type of `STRING`.

Mapping between MySQL/TDSQL original type and `DataType` is as shown below (the `MYSQL_TYPE_INT8/16/24/32/64` modified by `UNSIGNED` is respectively mapped to `UINT8/16/32/32/64`):

### Note

`DATE`, `TIME`, and `DATETIME` types don't support time zone.

The `TIMESTAMP` type supports time zone. Fields of this type will have their current time zone converted to Universal Time Coordinated (UTC) for storage, and vice versa for query.

The `MYSQL_TYPE_TIMESTAMP` and `MYSQL_TYPE_TIMESTAMP_NEW` fields carry the time zone information, which you can convert on your own when consuming data. For example, the format of the time data output by DTS is a string with time zone, such as `2021-05-17 07:22:42 +00:00`, where `+00:00` indicates the UTC time. You need to take into account the time zone information when parsing and converting the data.

MySQL/TDSQL Field Type	Protobuf DataType Value
<code>MYSQL_TYPE_NULL</code>	<code>NIL</code>
<code>MYSQL_TYPE_INT8</code>	<code>INT8</code>
<code>MYSQL_TYPE_INT16</code>	<code>INT16</code>
<code>MYSQL_TYPE_INT24</code>	<code>INT32</code>
<code>MYSQL_TYPE_INT32</code>	<code>INT32</code>
<code>MYSQL_TYPE_INT64</code>	<code>INT64</code>
<code>MYSQL_TYPE_BIT</code>	<code>INT64</code>
<code>MYSQL_TYPE_YEAR</code>	<code>INT64</code>
<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT32</code>
<code>MYSQL_TYPE_DOUBLE</code>	<code>FLOAT64</code>
<code>MYSQL_TYPE_VARCHAR</code>	<code>STRING</code>

MYSQL_TYPE_STRING	STRING
MYSQL_TYPE_VAR_STRING	STRING
MYSQL_TYPE_TIMESTAMP	STRING
MYSQL_TYPE_DATE	STRING
MYSQL_TYPE_TIME	STRING
MYSQL_TYPE_DATETIME	STRING
MYSQL_TYPE_TIMESTAMP_NEW	STRING
MYSQL_TYPE_DATE_NEW	STRING
MYSQL_TYPE_TIME_NEW	STRING
MYSQL_TYPE_DATETIME_NEW	STRING
MYSQL_TYPE_ENUM	STRING
MYSQL_TYPE_SET	STRING
MYSQL_TYPE_DECIMAL	DECIMAL
MYSQL_TYPE_DECIMAL_NEW	DECIMAL
MYSQL_TYPE_JSON	BYTES
MYSQL_TYPE_BLOB	BYTES
MYSQL_TYPE_TINY_BLOB	BYTES
MYSQL_TYPE_MEDIUM_BLOB	BYTES
MYSQL_TYPE_LONG_BLOB	BYTES
MYSQL_TYPE_GEOMETRY	BYTES

# Avro Demo Description

Last updated : 2024-07-08 16:52:02

## Key Logic Description

Files in the demo are as described below, with the Java demo as an example.

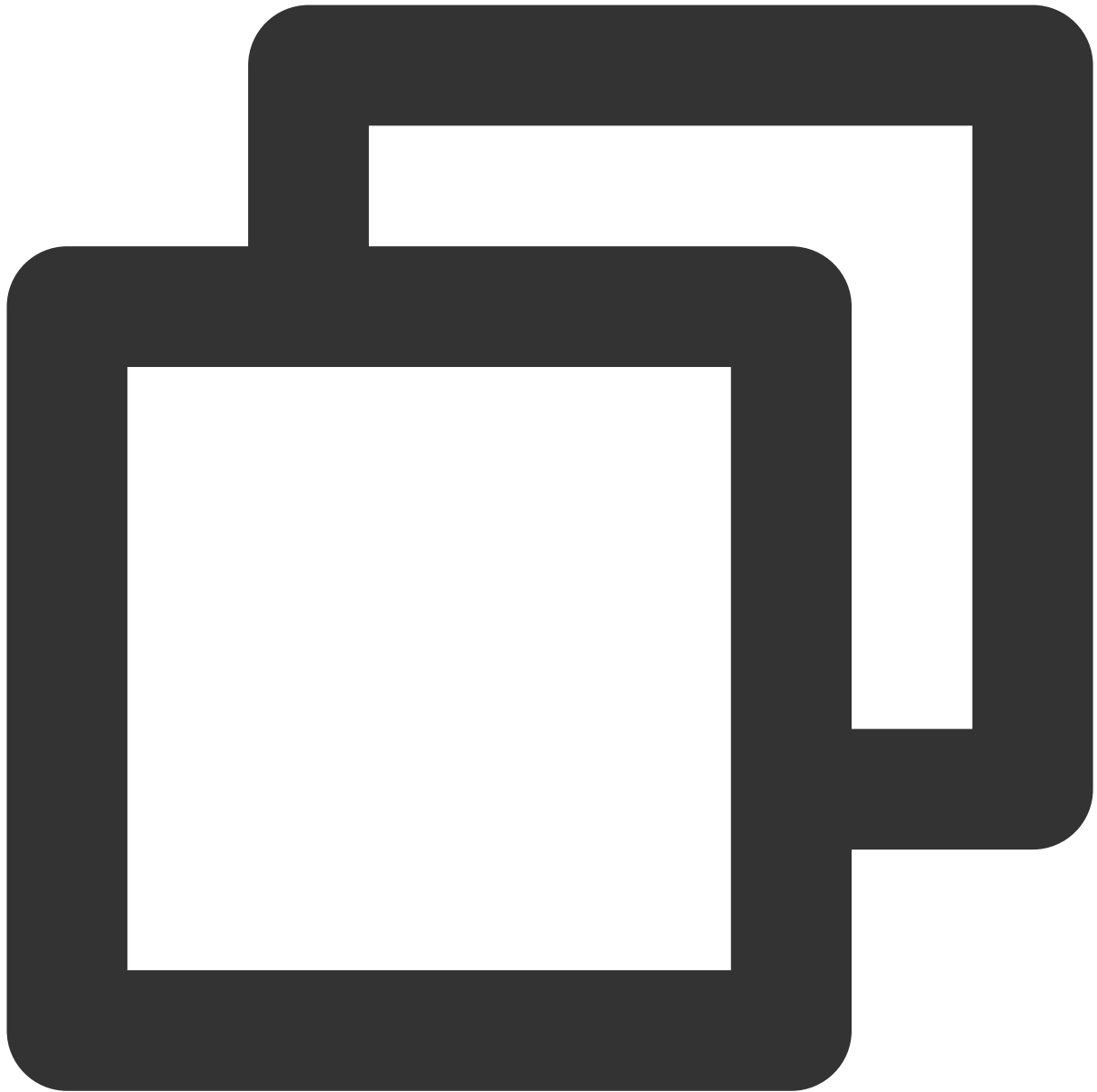
`consumerDemo-avro-java\src\main\resources\avro-tools-1.8.2.jar` : The tool used to generate Avro protocol code.

`consumerDemo-avro-java\src\main\java\com\tencent\subscribe\avro` : The directory of the Avro tool-generated code.

`consumerDemo-avro-java\src\main\resources\Record.avsc` : The protocol definition file.

14 structures (also known as schemas in Avro) are defined in `Record.avsc` . The main data structure is `record` , which is used to represent a data record in binlog. The record structure is as follows. Other data structures can be viewed in `Record.avsc` .





```
{
  "namespace": "com.tencent.subscribe.avro",    // The last schema in `Record.avsc`
  "type": "record",
  "name": "Record",    // `name` is displayed as `Record`, indicating the format
  "fields": [
    {
      "name": "id",    // `id` indicates a globally incremental ID. More record
      "type": "long",
      "doc": "unique id of this record in the whole stream"
    },
    {
```

```

    "name": "version",    // `version` indicates the protocol version.
    "type": "int",
    "doc": "protocol version"
  },
  {
    "name": "messageType",    // Message type
    "aliases": [
      "operation"
    ],
    "type": {
      "namespace": "com.tencent.subscribe.avro",
      "name": "MessageType",
      "type": "enum",
      "symbols": [
        "INSERT",
        "UPDATE",
        "DELETE",
        "DDL",
        "BEGIN",
        "COMMIT",
        "HEARTBEAT",
        "CHECKPOINT",
        "ROLLBACK"
      ]
    }
  },
  {
    .....
  },
}

```

Fields in a record are as explained below:

Field Name in Record	Description
id	The globally incremental ID
version	The protocol version, which is v1 currently.
messageType	The message type. Enumerated values: <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code> , <code>DDL</code> , <code>BEGIN</code> , <code>COMMIT</code> , <code>HEARTBEAT</code> , <code>CHECKPOINT</code> .
fileName	The name of the binlog file where the current record is located
position	The end offset of the current record in the binlog in the format of <code>End_log_pos@binlog file number</code> . For example, if the current record is in file <code>mysql-bin.000004</code> and the end offset is 2196, then the value of this parameter will be <code>2196@4</code> .

safePosition	The start offset of the current transaction in the binlog, which is in the same format as described above.
timestamp	The time when the data was written to the binlog, which is a UNIX timestamp in seconds.
gtid	The current GTID, such as c7c98333-6006-11ed-bfc9-b8cef6e1a231:9.
transactionId	The transaction ID, which is generated only for COMMIT events.
serverId	The server ID of the source database, which can be viewed by running <code>SHOW VARIABLES LIKE 'server_id' .</code>
threadId	The ID of the session that committed the current transaction, which can be viewed by running <code>SHOW processlist; .</code>
sourceType	The source database type, which currently can only be MySQL.
sourceVersion	The source database version, which can be viewed by running: <code>select version(); .</code>
schemaName	Database name
tableName	Table name
objectName	Format: Database name.table name
columns	The definitions of columns in the table
oldColumns	The data of the row before DML execution. If the message is an INSERT message, the array will be null. There are 12 element types in the array: Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, inaryGeometry, TextGeometry, BinaryObject, TextObject, and EmptyObject. For more information, see the demo.
newColumns	The data of the row after DML execution. If the message is a DELETE message, the array will be null. There are 12 element types in the array: Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, inaryGeometry, TextGeometry, BinaryObject, TextObject, and EmptyObject. For more information, see the demo.
sql	The DDL SQL statement
executionTime	The DDL execution duration in seconds
heartbeatTimestamp	The timestamp of the heartbeat message in seconds. This field is present only for heartbeat messages.

syncedGtid	The collection of GTIDs parsed by DTS in the format of <code>c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13</code>
fakeGtid	Whether the current GTID is forged. If <code>gtid_mode</code> is not enabled, DTS will forge a GTID.
pkNames	If the table in the source database has a primary key, this parameter will be carried in the DML message; otherwise, it will not be carried.
readerTimestamp	The time when DTS processed the current data record, which is a UNIX timestamp in milliseconds.
tags	The <code>status_vars</code> in <code>QueryEvent</code> . For more information, see <a href="#">QueryEvent</a> .
total	The total number of message segments if the message is segmented. This field is invalid on the current version ( <code>version=1</code> ) and is reserved for future use.
index	The index of a message segment if the message is segmented. This field is invalid on the current version ( <code>version=1</code> ) and is reserved for future use.

The field describing column attributes in a record is `Field`, including the following four attributes:

name: The column name.

dataTypeNumber: The type of the data recorded in the binlog. For values, see [MySQL source code documentation](#).

isKey: Whether the current key is the primary key.

originalType: The type defined in DDL.

## Database Field Mappings

The following lists the mappings between database (such as MySQL) field types and data types defined in the Avro protocol.

Type in MySQL	Corresponding Type in Avro
MYSQL_TYPE_NULL	EmptyObject
MYSQL_TYPE_INT8	Integer
MYSQL_TYPE_INT16	Integer
MYSQL_TYPE_INT24	Integer
MYSQL_TYPE_INT32	Integer

MYSQL_TYPE_INT64	Integer
MYSQL_TYPE_BIT	Integer
MYSQL_TYPE_YEAR	DateTime
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Float
MYSQL_TYPE_VARCHAR	Character
MYSQL_TYPE_STRING	Character. If the original type is binary, this type will correspond to BinaryObject.
MYSQL_TYPE_VAR_STRING	Character. If the original type is varbinary, this type will correspond to BinaryObject.
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_DATE	DateTime
MYSQL_TYPE_TIME	DateTime
MYSQL_TYPE_DATETIME	DateTime
MYSQL_TYPE_TIMESTAMP_NEW	Timestamp
MYSQL_TYPE_DATE_NEW	DateTime
MYSQL_TYPE_TIME_NEW	DateTime
MYSQL_TYPE_DATETIME_NEW	DateTime
MYSQL_TYPE_ENUM	TextObject
MYSQL_TYPE_SET	TextObject
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_DECIMAL_NEW	Decimal
MYSQL_TYPE_JSON	TextObject
MYSQL_TYPE_BLOB	BinaryObject
MYSQL_TYPE_TINY_BLOB	BinaryObject
MYSQL_TYPE_MEDIUM_BLOB	BinaryObject

MYSQL_TYPE_LONG_BLOB	BinaryObject
MYSQL_TYPE_GEOMETRY	BinaryObject

# JSON Demo Description

Last updated : 2024-07-08 16:52:02

The demo for each programming language uses JSON for serialization and contains a `Record` definition file.

In the demo for Java, the path of the definition file is `consumerDemo-json-java\\src\\main\\java\\json\\FlatRecord.java`.

## Type of Field in Record

Field Name in Record	Description
id	The globally incremental ID
version	The protocol version, which is v1 currently.
messageType	The message type. Enumerated values: "INSERT", "UPDATE", "DELETE", "DDL", "BEGIN", "COMMIT", "HEARTBEAT", "CHECKPOINT".
fileName	The name of the binlog file where the current record is located
position	The end offset of the current record in the binlog in the format of <code>End_log_pos@binlog file number</code> . For example, if the current record is in file <code>mysql-bin.000004</code> and the end offset is 2196, then the value of this parameter will be <code>2196@4</code> .
safePosition	The start offset of the current transaction in the binlog, which is in the same format as described above.
timestamp	The time when the data was written to the binlog, which is a UNIX timestamp in seconds.
gtid	The current GTID, such as c7c98333-6006-11ed-bfc9-b8cef6e1a231:9.
transactionId	The transaction ID, which is generated only for COMMIT events.
serverId	The server ID of the source database, which can be viewed by running <code>SHOW VARIABLES LIKE 'server_id'</code> .
threadId	The ID of the session that committed the current transaction, which can be viewed by running <code>SHOW processlist;</code> .
sourceType	The source database type, which currently can only be MySQL.

sourceVersion	The source database version, which can be viewed by running: <code>select version();</code>
schemaName	Database name
tableName	Table name
objectName	Format: Database name.table name
columns	The definitions of columns in the table
oldColumns	The data of the row before DML execution. If the message is an INSERT message, the array will be null.
newColumns	The data of the row after DML execution. If the message is a DELETE message, the array will be null.
sql	The DDL SQL statement
executionTime	The DDL execution duration in seconds
heartbeatTimestamp	The timestamp of the heartbeat message in seconds. This field is present only for heartbeat messages.
syncedGtid	The collection of GTIDs parsed by DTS in the format of <code>c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13</code>
fakeGtid	Whether the current GTID is forged. If <code>gtid_mode</code> is not enabled, DTS will forge a GTID.
pkNames	If the table in the source database has a primary key, this parameter will be carried in the DML message; otherwise, it will not be carried.
readerTimestamp	The time when DTS processed the current data record, which is a UNIX timestamp in milliseconds.
tags	The <code>status_vars</code> in <code>QueryEvent</code> . For more information, see <a href="#">QueryEvent</a> .
total	The total number of message segments if the message is segmented. This field is invalid on the current version (version=1) and is reserved for extension.
index	The index of message segments if the message is segmented. This field is invalid on the current version (version=1) and is reserved for extension.

## MySQL Column Attribute in Record



name: The column name.

dataTypeName: The type of the data recorded in the binlog. For values, see [MySQL source code documentation](#).

isKey: Whether the current key is the primary key.

originalType: The type defined in DDL.

## MySQL Data Type Conversion Logic

In the JSON protocol, all MySQL data types are converted to strings.

String types such as `varchar` are all converted to UTF-8 encoding.

Numeric types are all converted to strings equal to the value, such as "3.0".

Time types are output in the format of `yyyy-dd-mm hh:MM:ss.milli`.

Timestamp types are output as the number of milliseconds.

Binary types such as `binary` and `blob` are output as strings equal to their hex values, such as "0xff".

# Consuming Data with Flink

## Consuming MySQL or TDSQL-C for MySQL Subscribed Data Using Flink

Last updated : 2024-07-08 16:52:02

### Overview

In data subscription (Kafka Edition, where the current Kafka Server version is 2.6.0), subscribed data in Avro format can be consumed by using a Flink client (only the DataStream API type). This document provides a demo for data consumption with flink-dts-connector.

#### Note:

Currently, data consumption over the Avro protocol is supported only for TencentDB for MySQL and TDSQL-C for MySQL.

### Prerequisites

1. You have created a data consumption task as instructed in [Creating Data Subscription Task](#).
2. You have created a consumer group as instructed in [Adding Consumer Group](#).
3. You have installed [Flink](#), and it can execute tasks normally.

### Notes

**The demo only prints out the consumed data and does not contain any usage instructions. You need to write your own data processing logic based on the demo.** You can also use Kafka clients in other languages to consume and parse data.

Currently, data subscription to Kafka for consumption can be implemented over the Tencent Cloud private network but not the public network. In addition, the subscribed database instance and the data consumer must be in the same region.

The Kafka built in DTS has a certain upper limit for processing individual messages. When a single row of data in the source database exceeds 10 MB, this row may be discarded.

### Downloading a consumption demo

Demo Language	TencentDB for MySQL and TDSQL-C MySQL
Java	<a href="#">Address</a>

## Instructions for the Java Flink demo

Compilation environment: Maven or Gradle and JDK 8. You can choose a desired package management tool. The following takes Maven as an example.

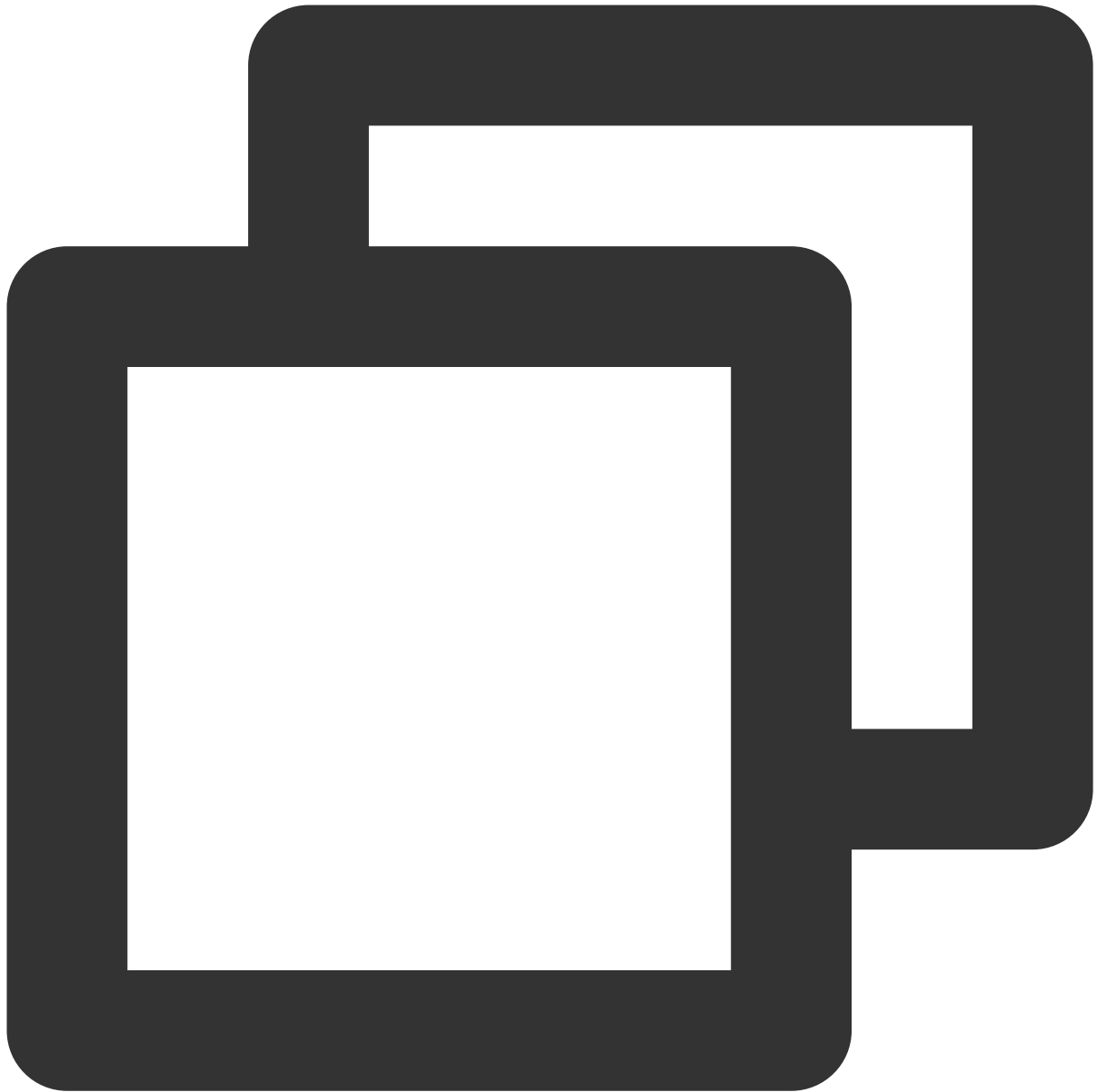
Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install JRE 8.

Directions:

1. Download the Java Flink demo and decompress it.
2. Access the decompressed directory. Maven model and pom.xml files are placed in the directory for your use as needed.

`java -jar avro-tools-1.8.2.jar compile -string schema Record.avsc` : Code generation path.

3. Modify the Flink version in the `pom.xml` file. The version in the following code must be the same as the Flink version you use.



```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-connector-kafka_${scala.binary.version}</artifactId>
  <version>1.13.6</version>
</dependency>
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
  <version>1.13.6</version>
  <scope>provided</scope>
</dependency>
```

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-avro</artifactId>
  <version>1.13.6</version>
</dependency>
```

4. Go to the directory where the pom file is located and package it with Maven or IEDA

Package with Maven by running `mvn clean package`.

5. For scenarios where the Flink client type is DataStream API, use Flink client commands to submit the task and start consumption.

```
./bin/flink run consumerDemo-avro-flink-1.0-SNAPSHOT.jar --brokers xxx --topic xxx
--group xxx --user xxx --password xxx -trans2sql
```

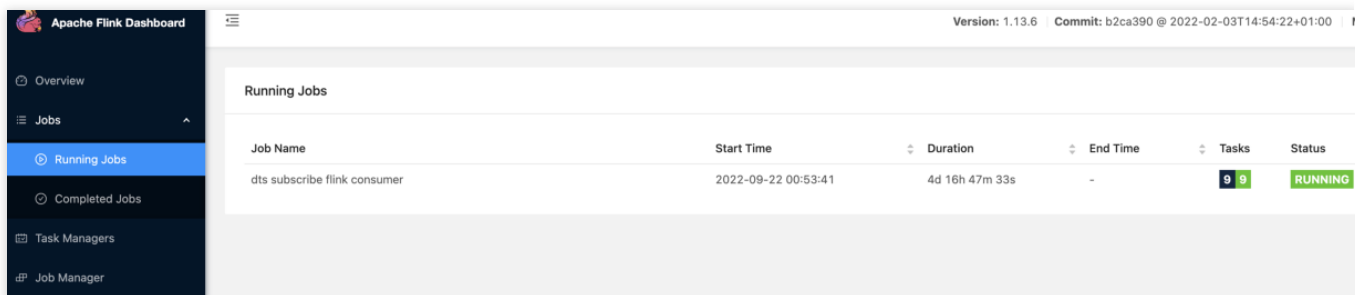
`broker` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement. In Java code, if this parameter is carried, the conversion will be enabled.

6. Observe consumption.

View running tasks.

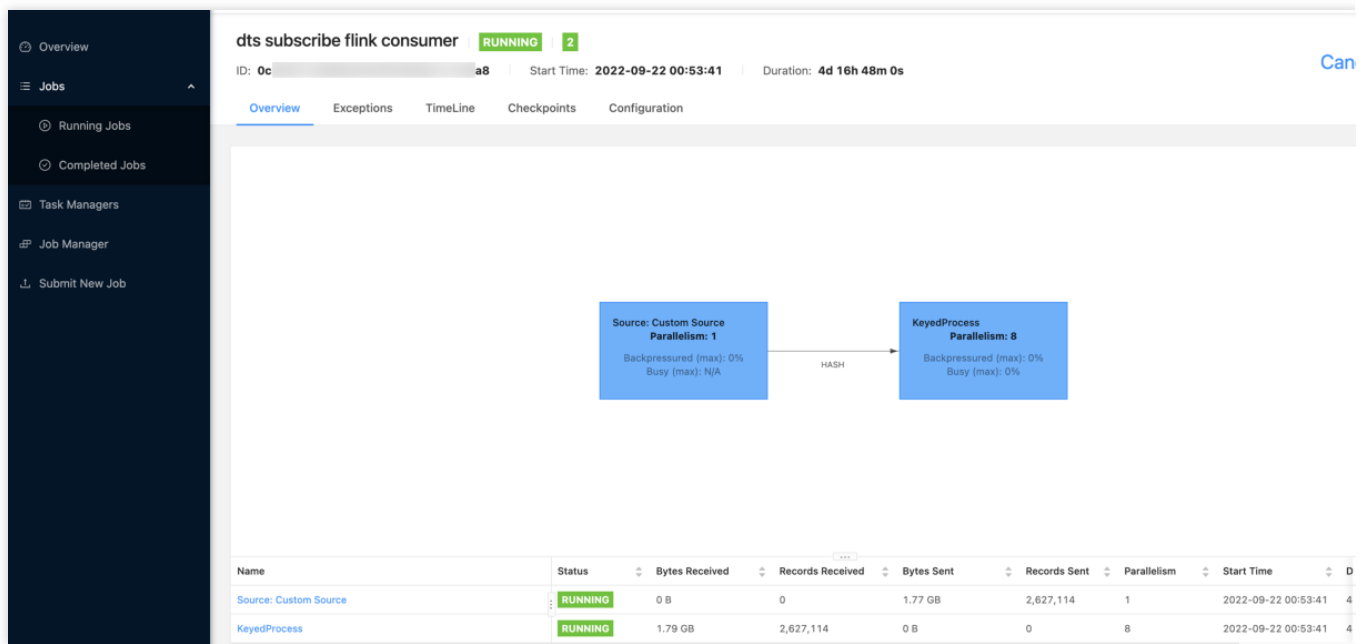


The screenshot shows the Apache Flink Dashboard interface. On the left is a dark sidebar with navigation links: Overview, Jobs (selected), Running Jobs, Completed Jobs, Task Managers, and Job Manager. The main panel is titled 'Running Jobs' and displays a table with the following data:

Job Name	Start Time	Duration	End Time	Tasks	Status
dts subscribe flink consumer	2022-09-22 00:53:41	4d 16h 47m 33s	-	9/9	RUNNING

The top right of the dashboard shows the version '1.13.6' and a commit hash 'b2ca390' with a timestamp '2022-02-03T14:54:22+01:00'.

View task details.



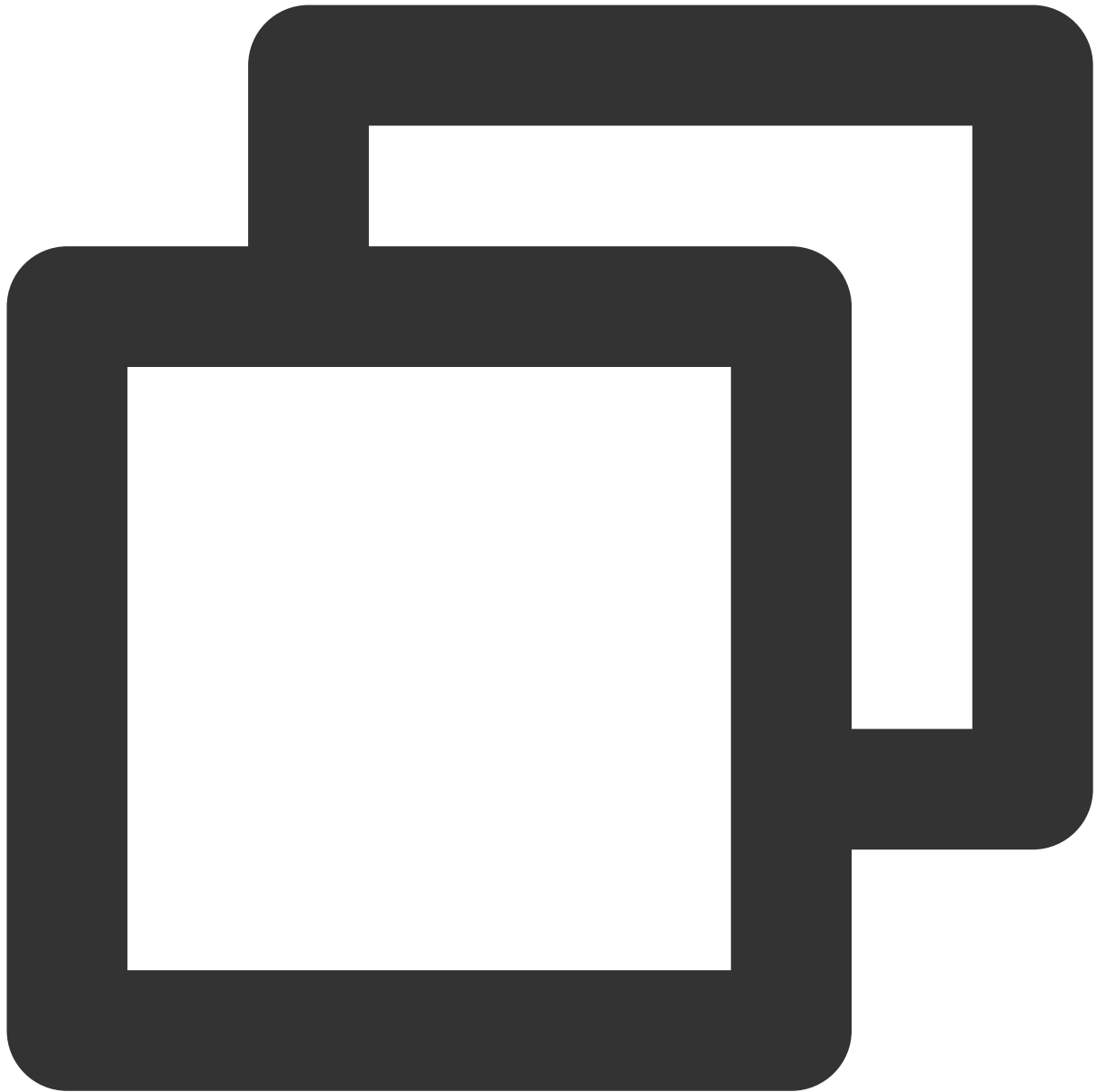
## Key demo logic description

Files in the demo are as described below:

`consumerDemo-avro-flink\src\main\resources\avro-tools-1.8.2.jar` : The tool used to generate Avro protocol code.

`consumerDemo-avro-flink\src\main\java\com\tencent\subscribe\avro` : The directory where the Avro tool generates code.

`consumerDemo-avro-flink\src\main\resources\Record.avsc` : The protocol definition file. 14 structures (called schemas in Avro) are defined in `Record.avsc` . The main data structure is record, which is used to represent a data record in binlog. The record structure is as follows. Other data structures can be viewed in `Record.avsc` .



```
{
  "namespace": "com.tencent.subscribe.avro",    // The last schema in `Record.avsc`
  "type": "record",
  "name": "Record",    // `name` is displayed as `Record`, indicating the format
  "fields": [
    {
      "name": "id",    // `id` indicates a globally incremental ID. More record
      "type": "long",
      "doc": "unique id of this record in the whole stream"
    },
    {
```

```

    "name": "version",    // `version` indicates the protocol version.
    "type": "int",
    "doc": "protocol version"
  },
  {
    "name": "messageType",    // Message type
    "aliases": [
      "operation"
    ],
    "type": {
      "namespace": "com.tencent.subscribe.avro",
      "name": "MessageType",
      "type": "enum",
      "symbols": [
        "INSERT",
        "UPDATE",
        "DELETE",
        "DDL",
        "BEGIN",
        "COMMIT",
        "HEARTBEAT",
        "CHECKPOINT",
        "ROLLBACK"
      ]
    }
  },
  {
    .....
  },
}

```

Fields in a record are as explained below:

Field	Description
id	The globally incremental ID.
version	The protocol version, which is v1 currently.
messageType	The message type. Enumerated values: "INSERT", "UPDATE", "DELETE", "DDL", "BEGIN", "COMMIT", "HEARTBEAT", "CHECKPOINT".
fileName	The name of the binlog file where the current record is located.
position	The end offset of the current record in the binlog in the format of <code>End_log_pos@binlog file number</code> . For example, if the current record is in file <code>mysql-bin.000004</code> and the end offset is 2196, then the value of this parameter will be <code>2196@4</code> .



safePosition	The start offset of the current transaction in the binlog, which is in the same format as described above.
timestamp	The time when the data was written to the binlog, which is a UNIX timestamp in seconds.
gtid	The current GTID, such as c7c98333-6006-11ed-bfc9-b8cef6e1a231:9.
transactionId	The transaction ID, which is generated only for COMMIT events.
serverId	The server ID of the source database, which can be viewed by running <code>SHOW VARIABLES LIKE 'server_id' .</code>
threadId	The ID of the session that committed the current transaction, which can be viewed by running <code>SHOW processlist; .</code>
sourceType	The source database type, which currently can only be MySQL.
sourceVersion	The source database version, which can be viewed by running: <code>select version(); .</code>
schemaName	The database name.
tableName	The table name.
objectName	Format: Database name.table name.
columns	The definitions of columns in the table.
oldColumns	The data of the row before DML execution. If the message is an INSERT message, the array will be null. There are 12 types of elements in the array, i.e., Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, and EmptyObject. For more information, see the definitions in the demo.
newColumns	The data of the row after DML execution. If the message is a DELETE message, the array will be null. There are 12 types of elements in the array, i.e., Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, and EmptyObject. For more information, see the definitions in the demo.
sql	The DDL SQL statement.
executionTime	The DDL execution duration in seconds.
heartbeatTimestamp	The timestamp of the heartbeat message in seconds, which is present only for

	heartbeat messages.
syncedGtid	The collection of GTIDs parsed by DTS in the format of <code>c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13</code> .
fakeGtid	Whether the current GTID is forged. If <code>gtid_mode</code> is not enabled, DTS will forge a GTID.
pkNames	If the table in the source database has a primary key, this parameter will be carried in the DML message; otherwise, it will not be carried.
readerTimestamp	The time when DTS processed the current data record, which is a UNIX timestamp in milliseconds.
tags	The <code>status_vars</code> in QueryEvent. For more information, see <a href="#">binary_log::Query_event Class Reference</a> .
total	The total number of message segments if the message is segmented. This field is invalid on the current version (version=1) and is reserved for extension.
index	The index of message segments if the message is segmented. This field is invalid on the current version (version=1) and is reserved for extension.

The field describing column attributes in a record is `Field` , including the following four attributes:

name: The column name.

dataTypeName: The type of the data recorded in the binlog. For values, see [COM\\_QUERY Response](#).

isKey: Whether the current key is the primary key.

originalType: The type defined in DDL.

## Database field mappings

The following lists the mappings between database (such as MySQL) field types and data types defined in the Avro protocol.

Type in MySQL	Corresponding Type in Avro
MYSQL_TYPE_NULL	EmptyObject
MYSQL_TYPE_INT8	Integer
MYSQL_TYPE_INT16	Integer
MYSQL_TYPE_INT24	Integer

MYSQL_TYPE_INT32	Integer
MYSQL_TYPE_INT64	Integer
MYSQL_TYPE_BIT	Integer
MYSQL_TYPE_YEAR	DateTime
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Float
MYSQL_TYPE_VARCHAR	Character
MYSQL_TYPE_STRING	Character. If the original type is binary, this type will correspond to BinaryObject.
MYSQL_TYPE_VAR_STRING	Character. If the original type is varbinary, this type will correspond to BinaryObject.
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_DATE	DateTime
MYSQL_TYPE_TIME	DateTime
MYSQL_TYPE_DATETIME	DateTime
MYSQL_TYPE_TIMESTAMP_NEW	Timestamp
MYSQL_TYPE_DATE_NEW	DateTime
MYSQL_TYPE_TIME_NEW	DateTime
MYSQL_TYPE_DATETIME_NEW	DateTime
MYSQL_TYPE_ENUM	TextObject
MYSQL_TYPE_SET	TextObject
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_DECIMAL_NEW	Decimal
MYSQL_TYPE_JSON	TextObject
MYSQL_TYPE_BLOB	BinaryObject
MYSQL_TYPE_TINY_BLOB	BinaryObject

---

MYSQL_TYPE_MEDIUM_BLOB	BinaryObject
MYSQL_TYPE_LONG_BLOB	BinaryObject
MYSQL_TYPE_GEOMETRY	BinaryObject

# Consuming TDSQL for MySQL Data with Flink

Last updated : 2024-07-08 16:52:02

## Overview

In data subscription (Kafka Edition, where the current Kafka Server version is 2.6.0), subscribed data in Protobuf format can be consumed by using a Flink client (only the DataStream API type). This document provides a demo for data consumption with **consumer-demo-tdsql-pb-flink** for you to quickly test the process of data consumption and understand the method of data parsing.

## Prerequisites

1. You have created a data consumption task as instructed in [Creating Data Subscription Task](#).
2. You have created a consumer group as instructed in [Creating Consumer Group](#).
3. You have installed [Flink](#), and it can execute tasks normally.

## Note

**The demo only prints out the consumed data and contains no instructions. You need to write your own data processing logic based on the demo.**

Currently, data subscription to Kafka for consumption **can only be implemented over the Tencent Cloud private network** but not the public network. In addition, the subscribed database instance and the data consumer must be in the same region.

In scenarios where only the specified databases and tables (part of the source instance objects) are subscribed and single-partition Kafka topics are used, only the data of the subscribed objects will be written to Kafka topics after DTS parses the incremental data. The data of non-subscription objects will be converted into empty transactions and then written to Kafka topics. Therefore, there are empty transactions during message consumption. The BEGIN and COMMIT messages in the empty transactions contain the GTID information, which can ensure the GTID continuity and integrity.

To ensure that data can be rewritten from where the task is paused, DTS adopts the checkpoint mechanism for data subscription. Specifically, when messages are written to Kafka topics, a checkpoint message is inserted every 10 seconds to mark the data sync offset. When the task is resumed after being interrupted, data can be rewritten from the

checkpointed offset. The consumer commits a consumption offset every time it encounters a checkpoint message so that the consumption offset can be updated timely.

## Downloading Consumption Demo

For the description of the logic and key parameters in the demo, see [Protobuf Demo Description \(Flink\)](#).

When configuring the subscription task, you can only select Protobuf as the format of the subscribed data for TDSQL for MySQL. As Protobuf adopts the binary format, the consumption efficiency is high. The following demo already contains the Protobuf protocol file, so you don't need to download it separately. If you [download](#) it on your own, use Protobuf 3.X for code generation to ensure that data structures are compatible.

Demo Language	Protobuf (TDSQL for MySQL)
Java	<a href="#">Address</a>

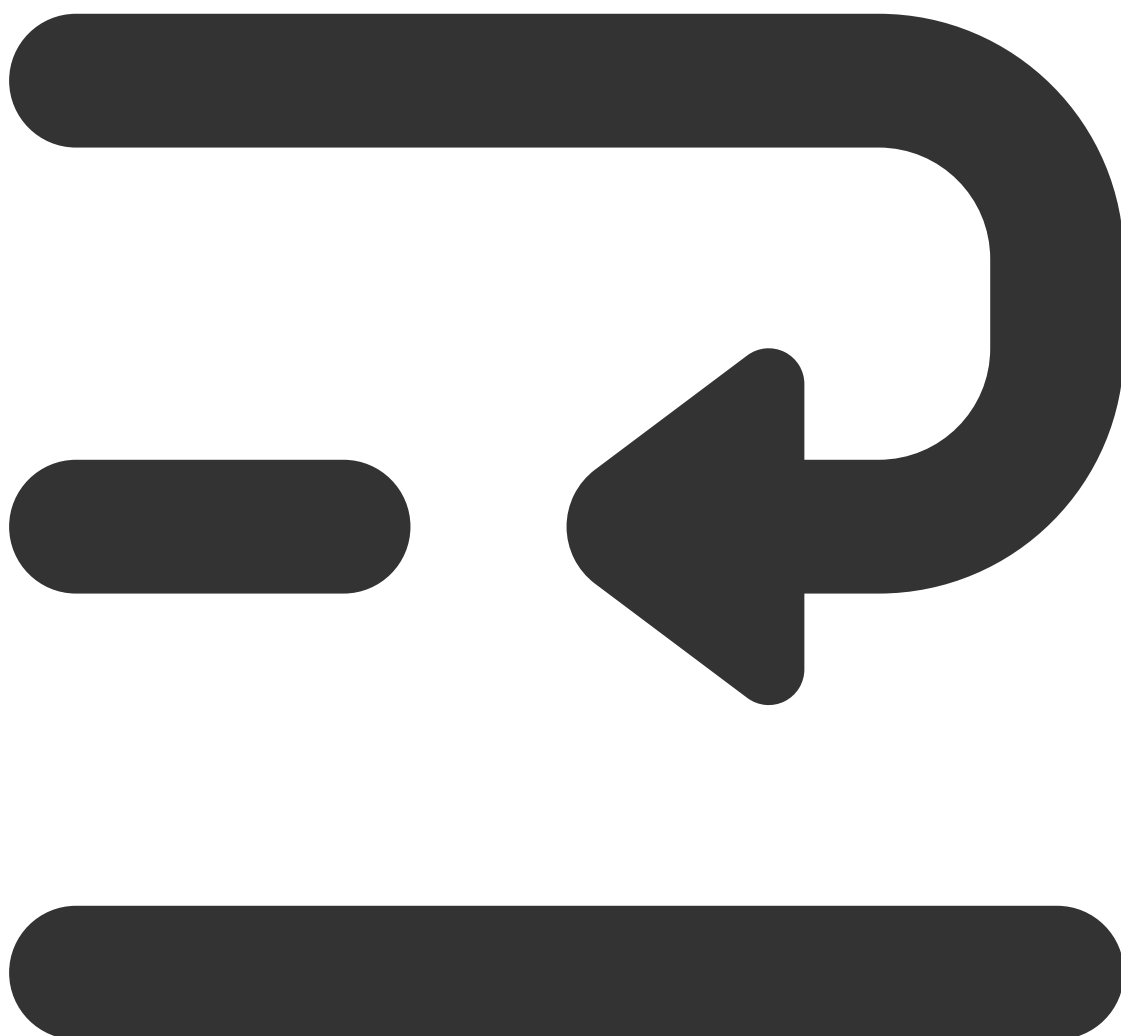
## Directions for the Java Flink Demo

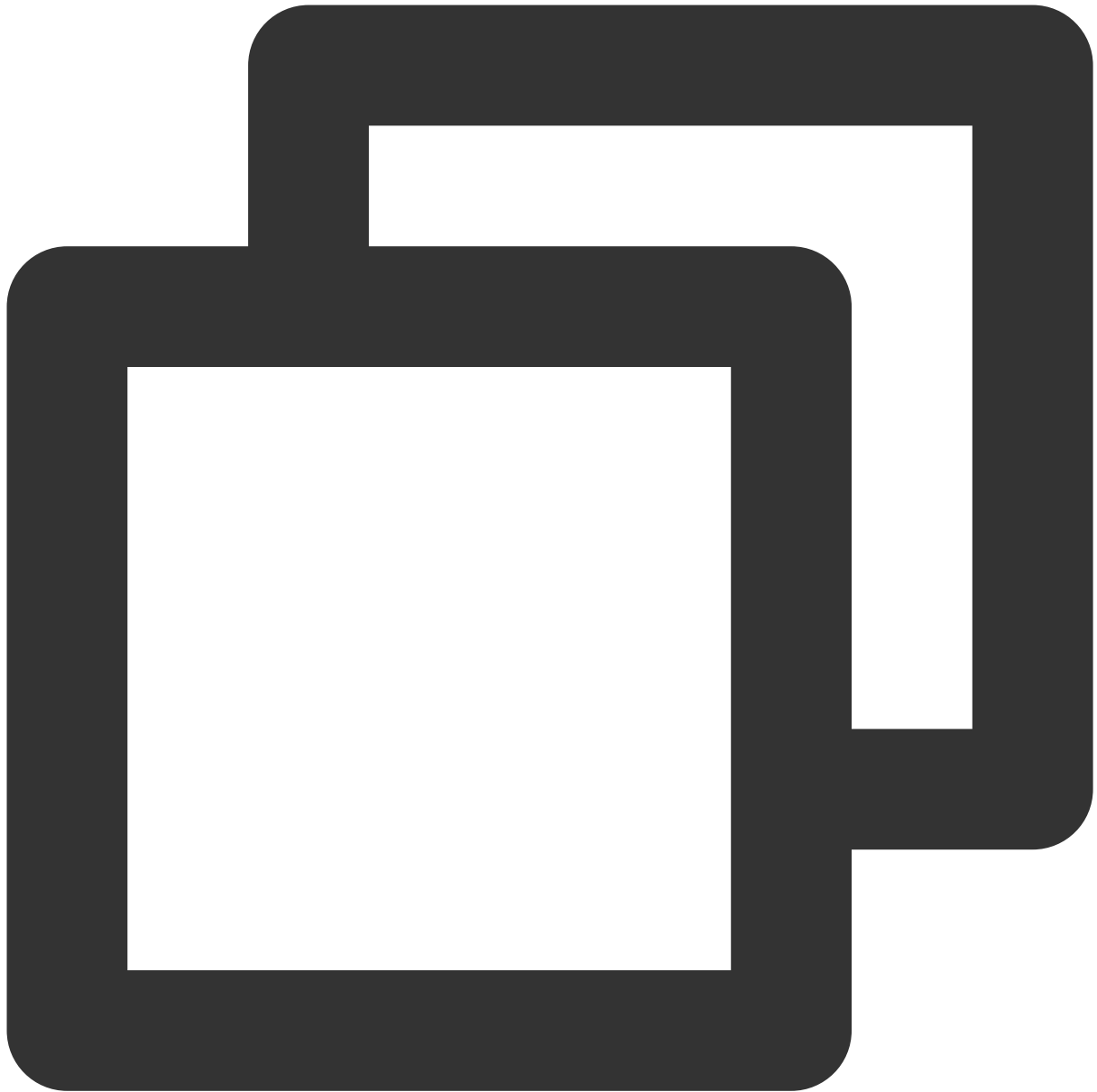
**Compiling environment:** Maven and JDK8. You can choose a desired package management tool. The following takes Maven as an example.

**Runtime environment:** Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install JRE8.

### Directions:

1. Download the **consumer-demo-tdsql-pb-flink.zip** file and unzip it.
2. Access the directory of the unzipped file. The `pom.xml` file has been placed under the directory for your convenience. You need to modify the Flink cluster version to the same as that specified in the `pom.xml` dependency.
3. The value of `${flink.version}` in the code below must be the same as the Flink cluster version.





```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-connector-kafka_${scala.binary.version}</artifactId>
  <version>${flink.version}</version>
```



```
<exclusions>
  <exclusion>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
  </exclusion>
</exclusions>
</dependency>

<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-clients_${scala.binary.version}</artifactId>
  <version>${flink.version}</version>
</dependency>
```

4. Go to the directory where the `pom.xml` file is located and package it with Maven or IEDA.

Package with Maven by running `mvn clean package`.

5. If the Flink client type is DataStream API, use Flink client commands to submit the job to the Flink cluster and start consumption.

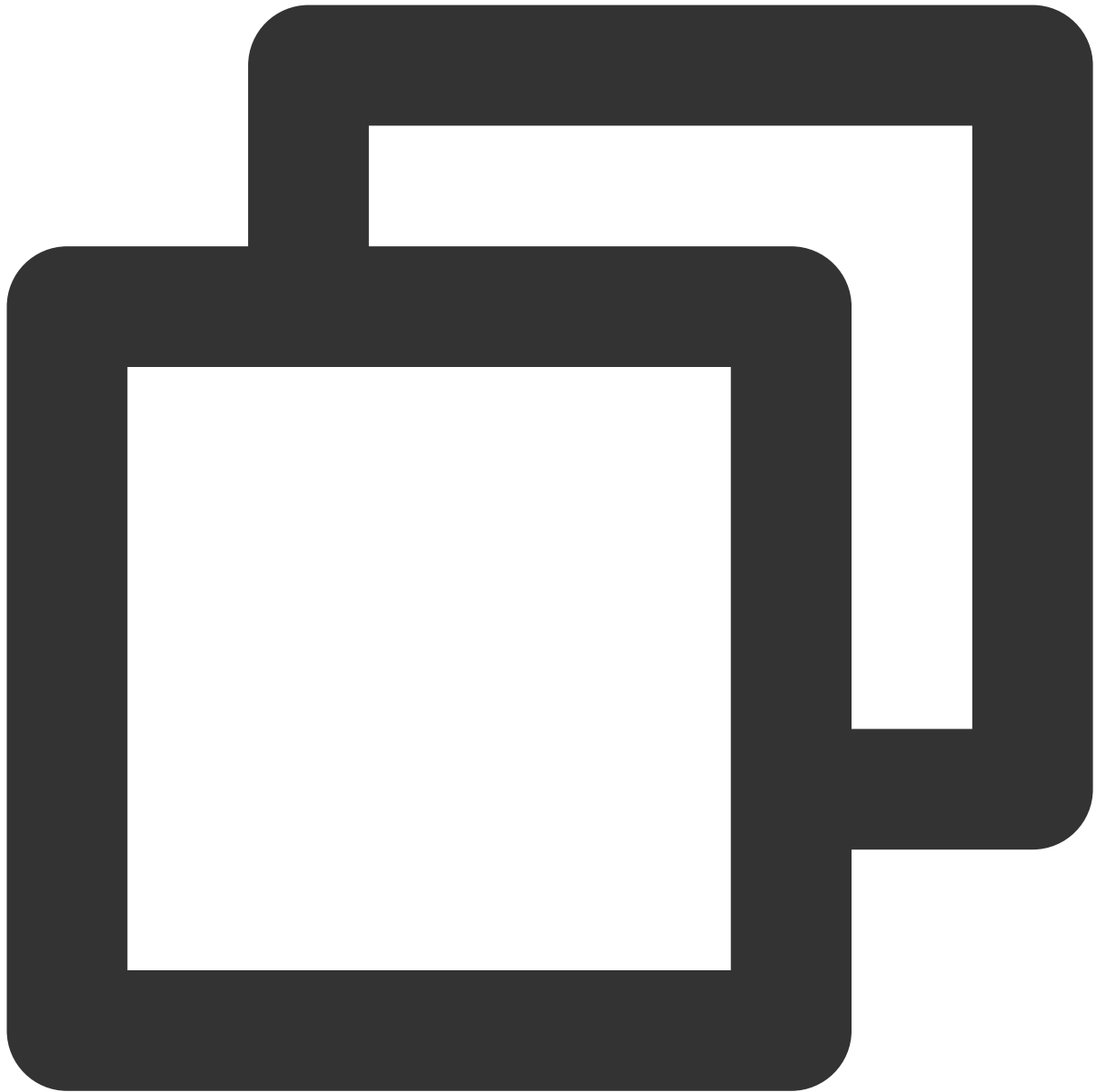
```
./bin/flink run consumer-demo-tdsql-pb-flink.jar --brokers xxx --topic xxx --group
xxx --user xxx --password xxx --trans2sql
```

`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement. In Java code, if this parameter is carried, the conversion will be enabled.

6. Execute DML statements in the source database.



```
CREATE TABLE `flink_test` (  
  `id` varchar(32) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,  
  `parent_id` varchar(32) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci DE  
  `user_id` bigint NOT NULL,  
  `type` int NOT NULL COMMENT '1: Expenditure 2: Income',  
  `create_time` timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),  
  PRIMARY KEY (`id`) USING BTREE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci ROW_FORMAT=DYNAM
```

7. Observe the consumption performed by the previously submitted job.

View the specific task logs on task managers.

View the specific Stdout information on task managers.

# Demo Description

## Avro Demo Description (Flink)

Last updated : 2024-07-08 16:52:02

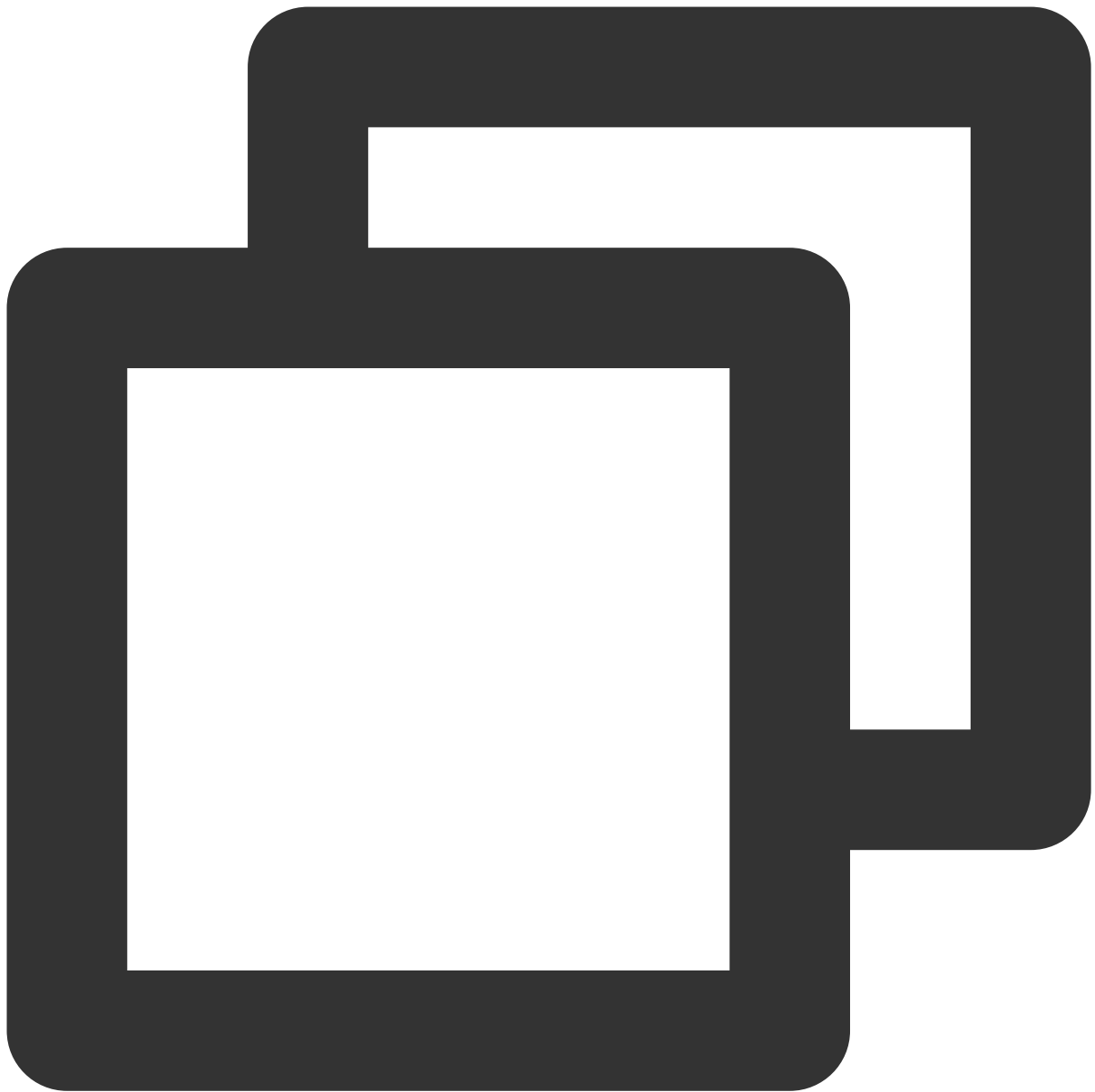
### Key Logic Description

Files in the demo are as described below:

`consumerDemo-avro-flink\\src\\main\\resources\\avro-tools-1.8.2.jar` : The tool used to generate Avro protocol code.

`consumerDemo-avro-flink\\src\\main\\java\\com\\tencent\\subscribe\\avro` : The directory of the Avro tool-generated code.

`consumerDemo-avro-flink\\src\\main\\resources\\Record.avsc` : The protocol definition file. 14 structures (also known as schemas in Avro) are defined in `Record.avsc` . The main data structure is `record` , which is used to represent a data record in binlog. The record structure is as follows. Other data structures can be viewed in `Record.avsc` .



```
{
  "namespace": "com.tencent.subscribe.avro",    // The last schema in `Record.avsc`
  "type": "record",
  "name": "Record",    // `name` is displayed as `Record`, indicating the format
  "fields": [
    {
      "name": "id",    // `id` indicates a globally incremental ID. More record
      "type": "long",
      "doc": "unique id of this record in the whole stream"
    },
    {
```

```

    "name": "version",    // `version` indicates the protocol version.
    "type": "int",
    "doc": "protocol version"
  },
  {
    "name": "messageType",    // Message type
    "aliases": [
      "operation"
    ],
    "type": {
      "namespace": "com.tencent.subscribe.avro",
      "name": "MessageType",
      "type": "enum",
      "symbols": [
        "INSERT",
        "UPDATE",
        "DELETE",
        "DDL",
        "BEGIN",
        "COMMIT",
        "HEARTBEAT",
        "CHECKPOINT",
        "ROLLBACK"
      ]
    }
  },
  {
    .....
  },
}

```

Fields in a record are as explained below:

Field Name in Record	Description
id	The globally incremental ID
version	The protocol version, which is v1 currently.
messageType	The message type. Enumerated values: INSERT , UPDATE , DELETE , DDL , BEGIN , COMMIT , HEARTBEAT , CHECKPOINT .
fileName	The name of the binlog file where the current record is located
position	The end offset of the current record in the binlog in the format of End_log_pos@binlog file number . For example, if the current record is in file mysql-bin.000004 and the end offset is 2196, then the value of this parameter will be 2196@4 .

safePosition	The start offset of the current transaction in the binlog, which is in the same format as described above.
timestamp	The time when the data was written to the binlog, which is a UNIX timestamp in seconds.
gtid	The current GTID, such as c7c98333-6006-11ed-bfc9-b8cef6e1a231:9.
transactionId	The transaction ID, which is generated only for COMMIT events.
serverId	The server ID of the source database, which can be viewed by running <code>SHOW VARIABLES LIKE 'server_id' .</code>
threadId	The ID of the session that committed the current transaction, which can be viewed by running <code>SHOW processlist; .</code>
sourceType	The source database type, which currently can only be MySQL.
sourceVersion	The source database version, which can be viewed by running: <code>select version(); .</code>
schemaName	Database name
tableName	Table name
objectName	Format: Database name.table name
columns	The definitions of columns in the table
oldColumns	The data of the row before DML execution. If the message is an INSERT message, the array will be null. There are 12 element types in the array: Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, and EmptyObject.
newColumns	The data of the row after DML execution. If the message is a DELETE message, the array will be null. There are 12 element types in the array: Integer, Character, Decimal, Float, Timestamp, DateTime, TimestampWithTimeZone, BinaryGeometry, TextGeometry, BinaryObject, TextObject, and EmptyObject.
sql	The DDL SQL statement
executionTime	The DDL execution duration in seconds
heartbeatTimestamp	The timestamp of the heartbeat message in seconds. This field is present only for heartbeat messages.
syncedGtid	The collection of GTIDs parsed by DTS in the format of <code>c7c98333-6006-11ed-bfc9-b8cef6e1a231:1-13 .</code>

fakeGtid	Whether the current GTID is forged. If <code>gtid_mode</code> is not enabled, DTS will forge a GTID.
pkNames	If the table in the source database has a primary key, this parameter will be carried in the DML message; otherwise, it will not be carried.
readerTimestamp	The time when DTS processed the current data record, which is a UNIX timestamp in milliseconds.
tags	The <code>status_vars</code> in <code>QueryEvent</code> . For more information, see <a href="#">QueryEvent</a> .
total	The total number of message segments if the message is segmented. This field is invalid on the current version (version=1) and is reserved for extension.
index	The index of message segments if the message is segmented. This field is invalid on the current version (version=1) and is reserved for extension.

The field describing column attributes in a record is `Field`, including the following four attributes:

name: The column name.

dataTypeNumber: The type of the data recorded in the binlog. For values, see [MySQL source code documentation](#).

isKey: Whether the current key is the primary key.

originalType: The type defined in DDL.

## Database Field Mappings

The following lists the mappings between database (such as MySQL) field types and data types defined in the Avro protocol.

Type in MySQL	Corresponding Type in Avro
MYSQL_TYPE_NULL	EmptyObject
MYSQL_TYPE_INT8	Integer
MYSQL_TYPE_INT16	Integer
MYSQL_TYPE_INT24	Integer
MYSQL_TYPE_INT32	Integer
MYSQL_TYPE_INT64	Integer
MYSQL_TYPE_BIT	Integer



MYSQL_TYPE_YEAR	DateTime
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Float
MYSQL_TYPE_VARCHAR	Character
MYSQL_TYPE_STRING	Character. If the original type is binary, this type will correspond to BinaryObject.
MYSQL_TYPE_VAR_STRING	Character. If the original type is varbinary, this type will correspond to BinaryObject.
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_DATE	DateTime
MYSQL_TYPE_TIME	DateTime
MYSQL_TYPE_DATETIME	DateTime
MYSQL_TYPE_TIMESTAMP_NEW	Timestamp
MYSQL_TYPE_DATE_NEW	DateTime
MYSQL_TYPE_TIME_NEW	DateTime
MYSQL_TYPE_DATETIME_NEW	DateTime
MYSQL_TYPE_ENUM	TextObject
MYSQL_TYPE_SET	TextObject
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_DECIMAL_NEW	Decimal
MYSQL_TYPE_JSON	TextObject
MYSQL_TYPE_BLOB	BinaryObject
MYSQL_TYPE_TINY_BLOB	BinaryObject
MYSQL_TYPE_MEDIUM_BLOB	BinaryObject
MYSQL_TYPE_LONG_BLOB	BinaryObject
MYSQL_TYPE_GEOMETRY	BinaryObject



# ProtoBuf Demo Description (Flink)

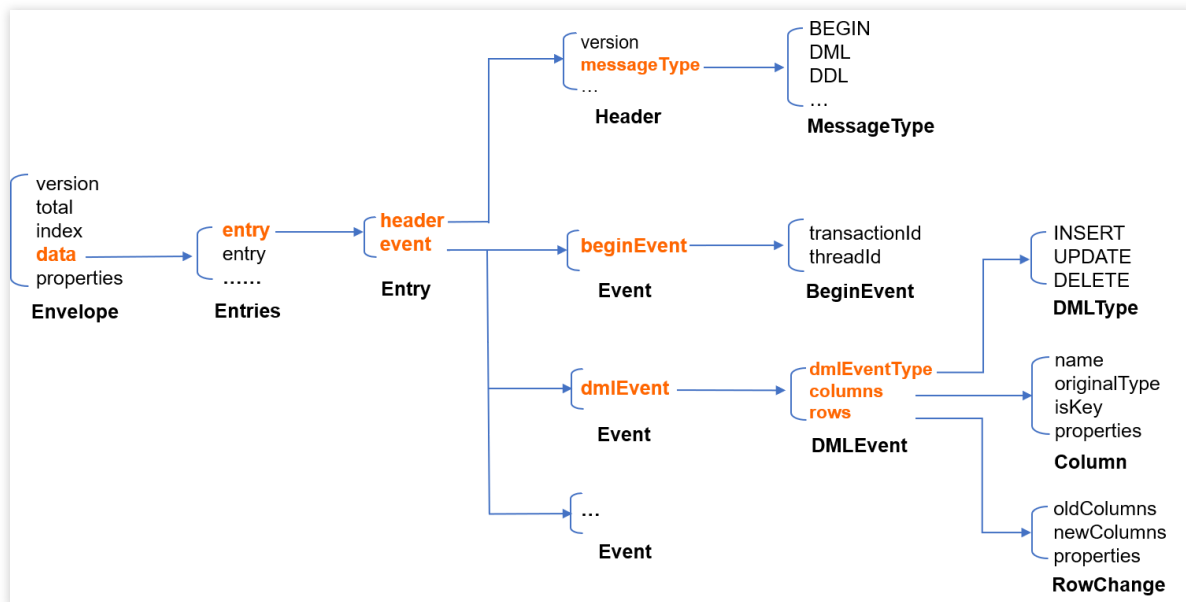
Last updated : 2024-07-08 16:52:02

## Key Logic Description

### Message production logic

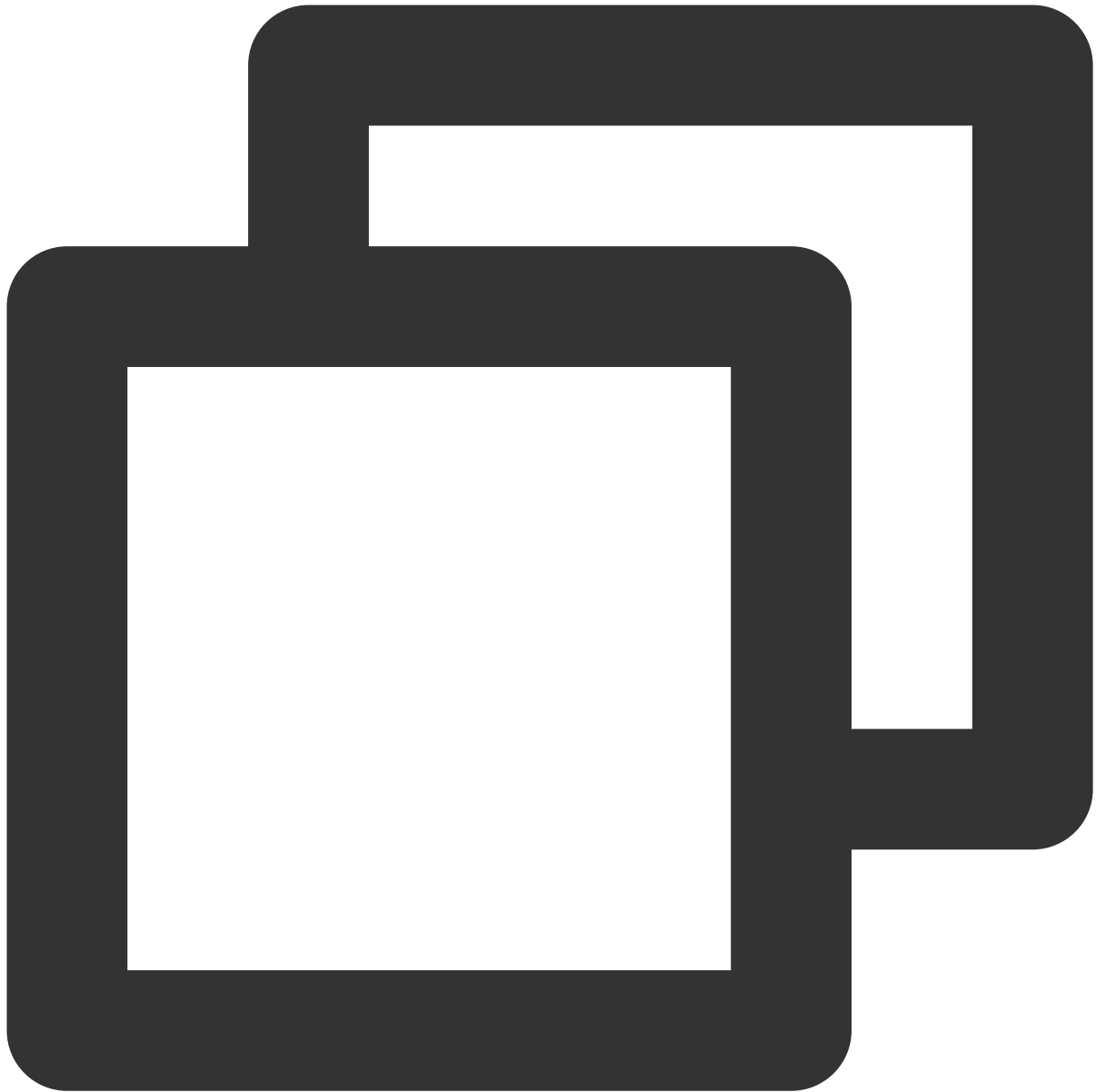
This section describes the message production logic to help you better understand the consumption logic.

The demo uses Protobuf for serialization and contains a `Protobuf` definition file. In the file, three key structures are defined as follows: `Envelope` is the final Kafka message structure; `Entry` is the structure of a single subscription event; `Entries` is the collection of `Entry`. Their relationship is shown below:



The production process is as follows:

1. Pull binlog messages and encode each binlog event into an `Entry`.



```
message Entry { // An `Entry` is the structure of an individual subscription event.
  Header header = 1; // The event header
  Event event   = 2; // The event body
}
```

```
message Header {
  int32      version      = 1; // The protocol version of the `Entry`
  SourceType sourceType   = 2; // The source database type, such as MySQL and O
  MessageType messageType = 3; // The message type, i.e., event type, such as B
  uint32 timestamp        = 4; // The event timestamp in the source binlog
}
```

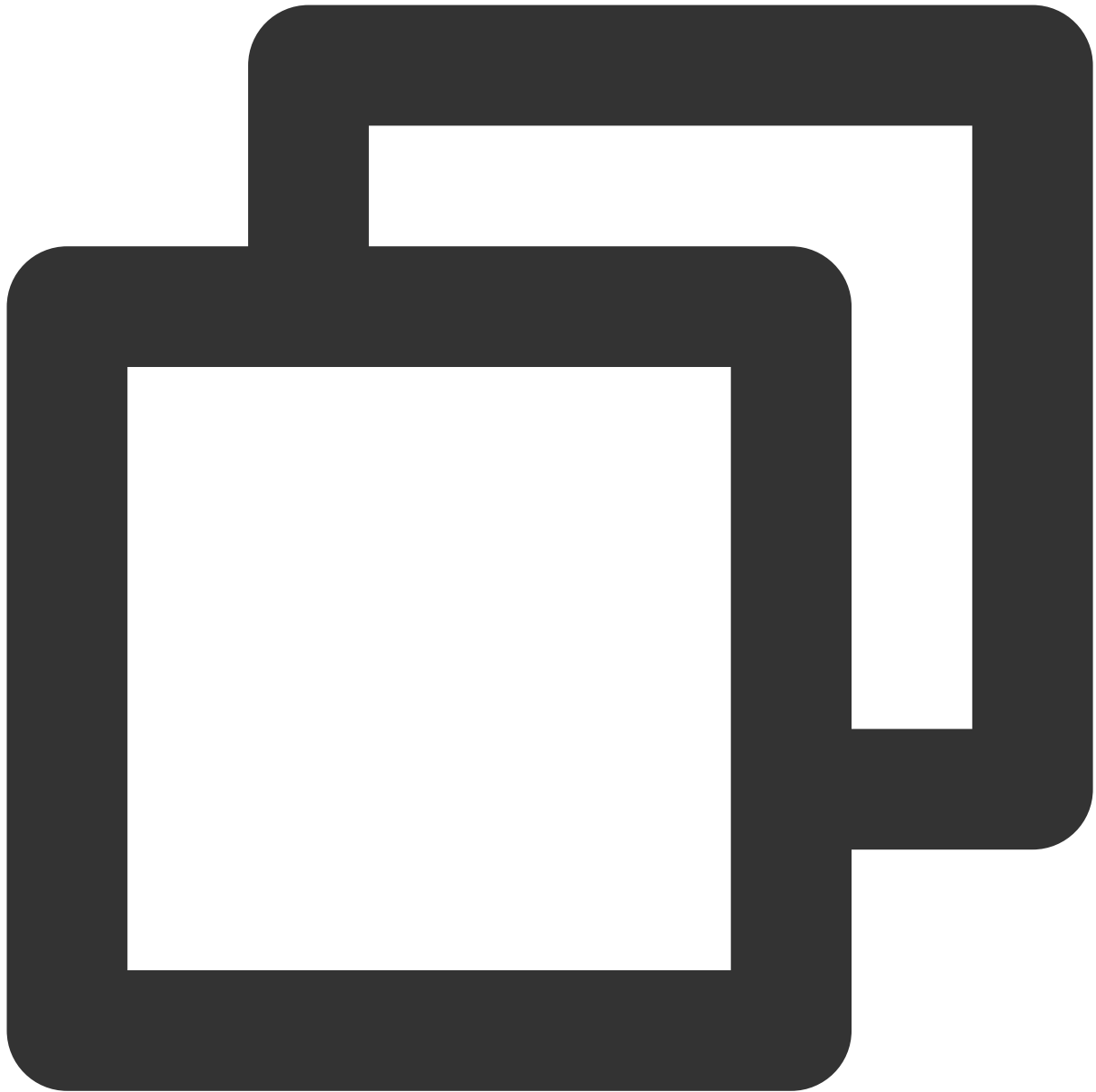
```

int64  serverId      = 5;    // The `serverId` of the source database
string fileName      = 6;    // The filename of the source binlog
uint64 position      = 7;    // The event offset in the source binlog file
string gtId          = 8;    // The GTID of the current transaction
string schemaName    = 9;    // The modified schema
string tableName     = 10;   // The modified table
uint64 seqId         = 11;   // The globally incremental serial number
uint64 eventIndex    = 12;   // If a large event is sharded, the shard number
bool   isLast        = 13;   // Whether the current shard is the last shard o
repeated KVPair properties = 15;
}

message Event {
  BeginEvent      beginEvent      = 1;  // The BEGIN event in the binlog
  DMLEvent        dmlEvent        = 2;  // The DML event in the binlog
  CommitEvent     commitEvent     = 3;  // The COMMIT event in the binlog
  DDLEvent        ddlEvent        = 4;  // The DDL event in the binlog
  RollbackEvent   rollbackEvent   = 5;  // The rollback event. This parameter is mea
  HeartbeatEvent  heartbeatEvent  = 6;  // The heartbeat event regularly sent by the
  CheckpointEvent checkpointEvent = 7;  // The checkpoint event added to the subscri
  repeated KVPair properties      = 15;
}

```

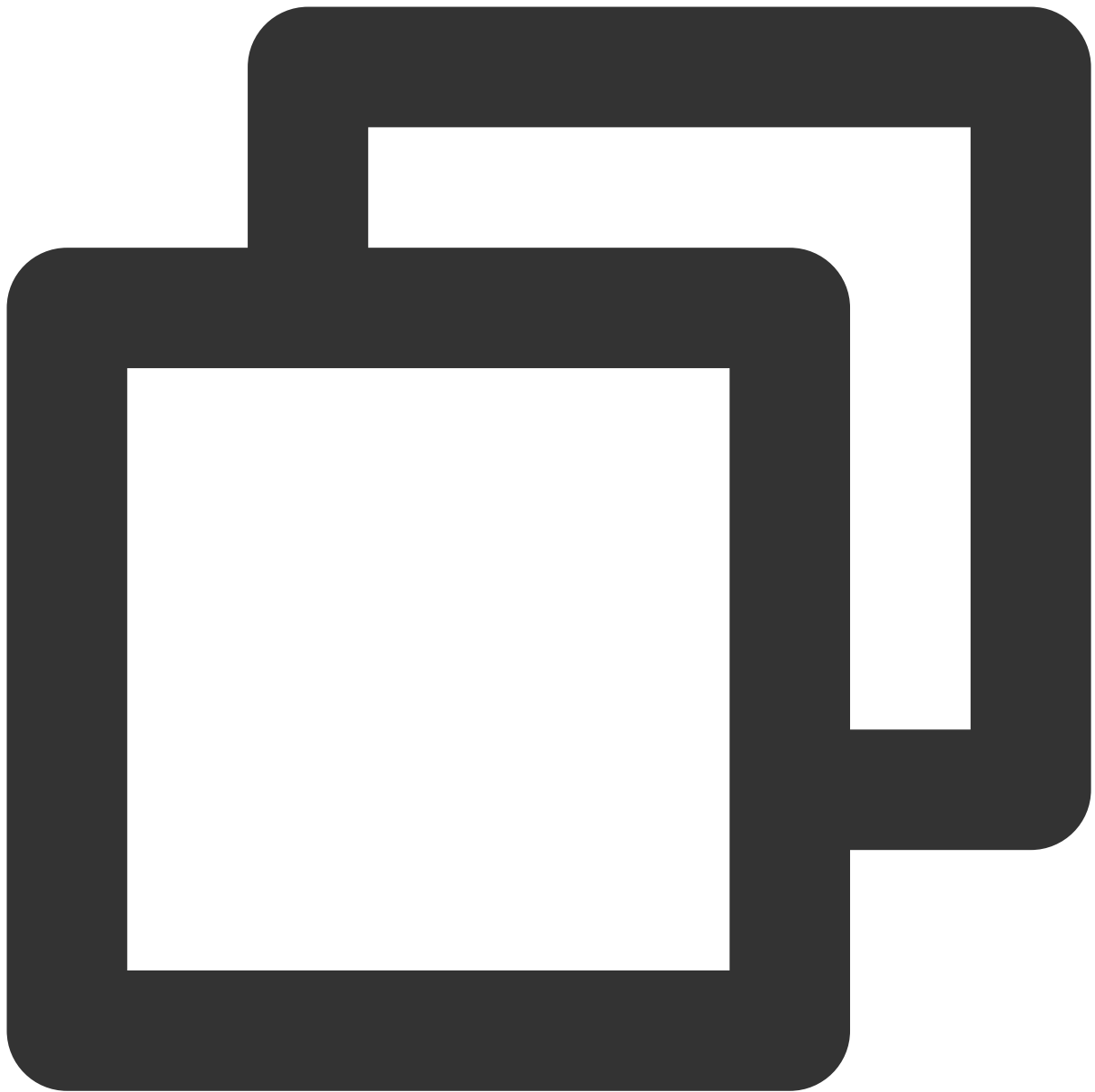
2. Multiple `Entry` structures are merged to reduce the number of messages, and the structure of binlog events becomes `Entries` after the merge. The `Entries.items` field refers to the `Entry` sequence list. The reasonable number of merged `Entry` structures should be smaller than that of a single Kafka message. If a single binlog event has exceeded the size limit, `Entry` structures will not be merged anymore, so there will be only one `Entry` in the `Entries` structure.



```
message Entries {  
    repeated Entry items = 1; // `Entry` list  
}
```

3. Encode `Entries` with Protobuf to generate a binary sequence.

4. Put the binary sequence in the `data` field of an `Envelope`. If a single binlog event is oversize, the binary sequence may exceed the size limit of a single Kafka message. In this case, you can separate the binary sequence into multiple segments and put each segment in an `Envelope`.



```
message Envelope {  
    int32  version                = 1; // The protocol version, which determines  
    uint32 total                  = 2;  
    uint32 index                  = 3;  
    bytes  data                   = 4; // Here, `version` is 1, indicating that  
    repeated KVPair properties    = 15;  
}
```

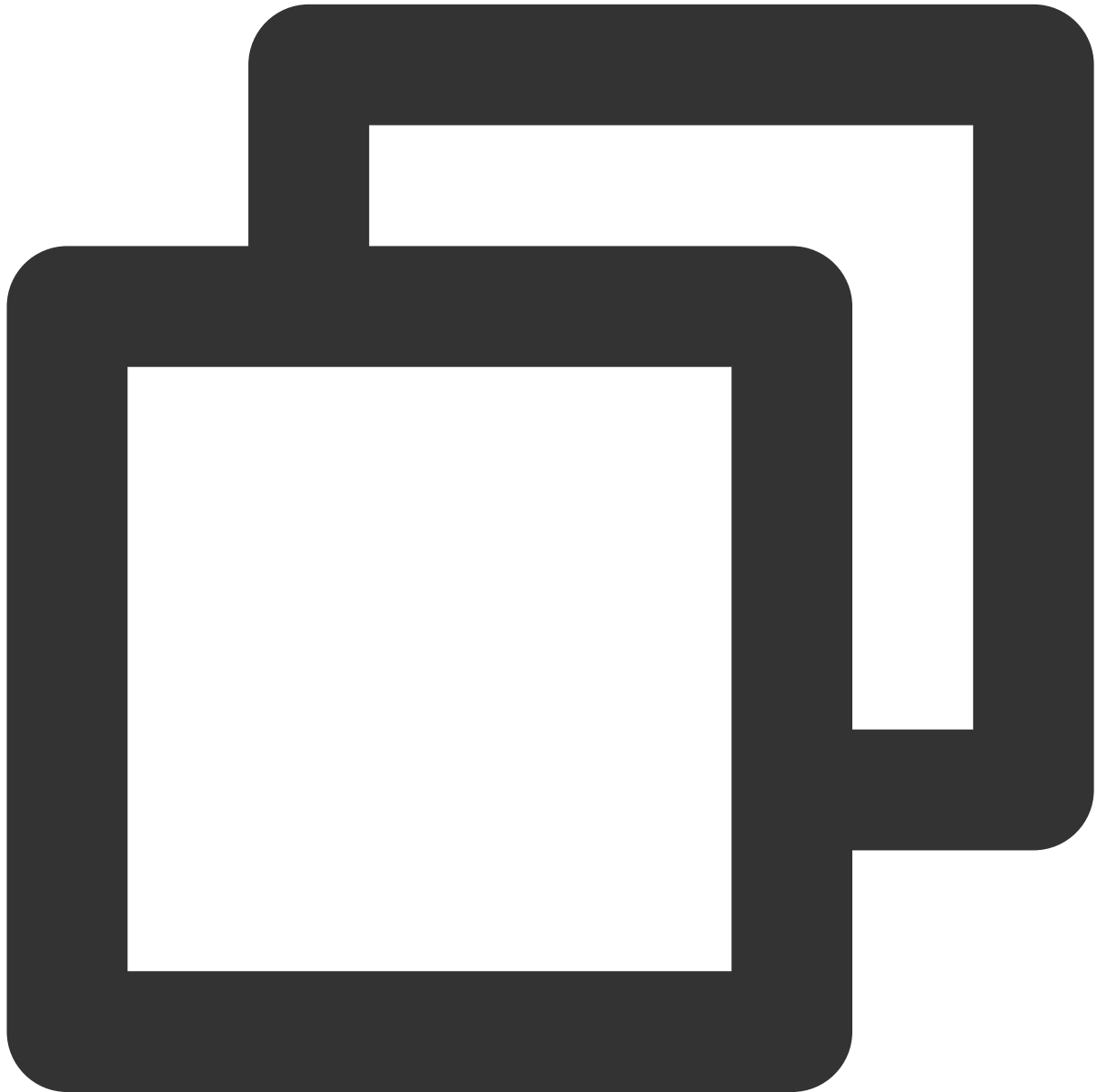
5. Encode one or multiple `Envelope` structures generated in the previous step in sequence and deliver the `Envelope` structures to Kafka partitions. Multiple `Envelope` structures in the same `Entries` are delivered

to the same partition in sequence.

## Message consumption logic

This section describes the message consumption logic.

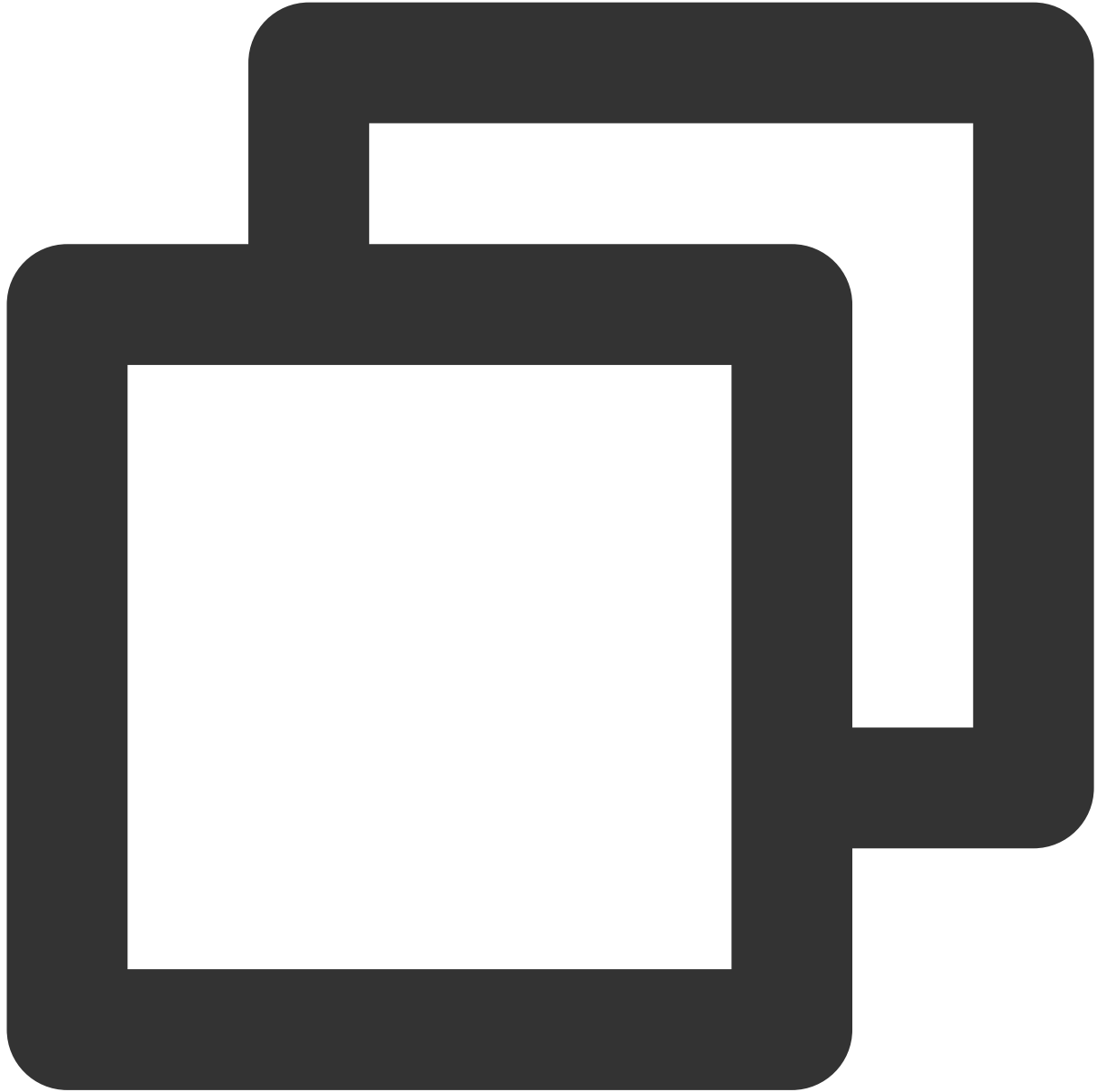
1. To use Flink to consume messages, you need to create a `FlinkKafkaConsumer`, specify a consumption topic, and customize a message deserializer based on the Protobuf protocol.



```
// Create a FlinkKafkaConsumer
FlinkKafkaConsumer<RecordMsgObject> consumer =
    new FlinkKafkaConsumer<>(topic, new DeserializeProtobufToRec
```



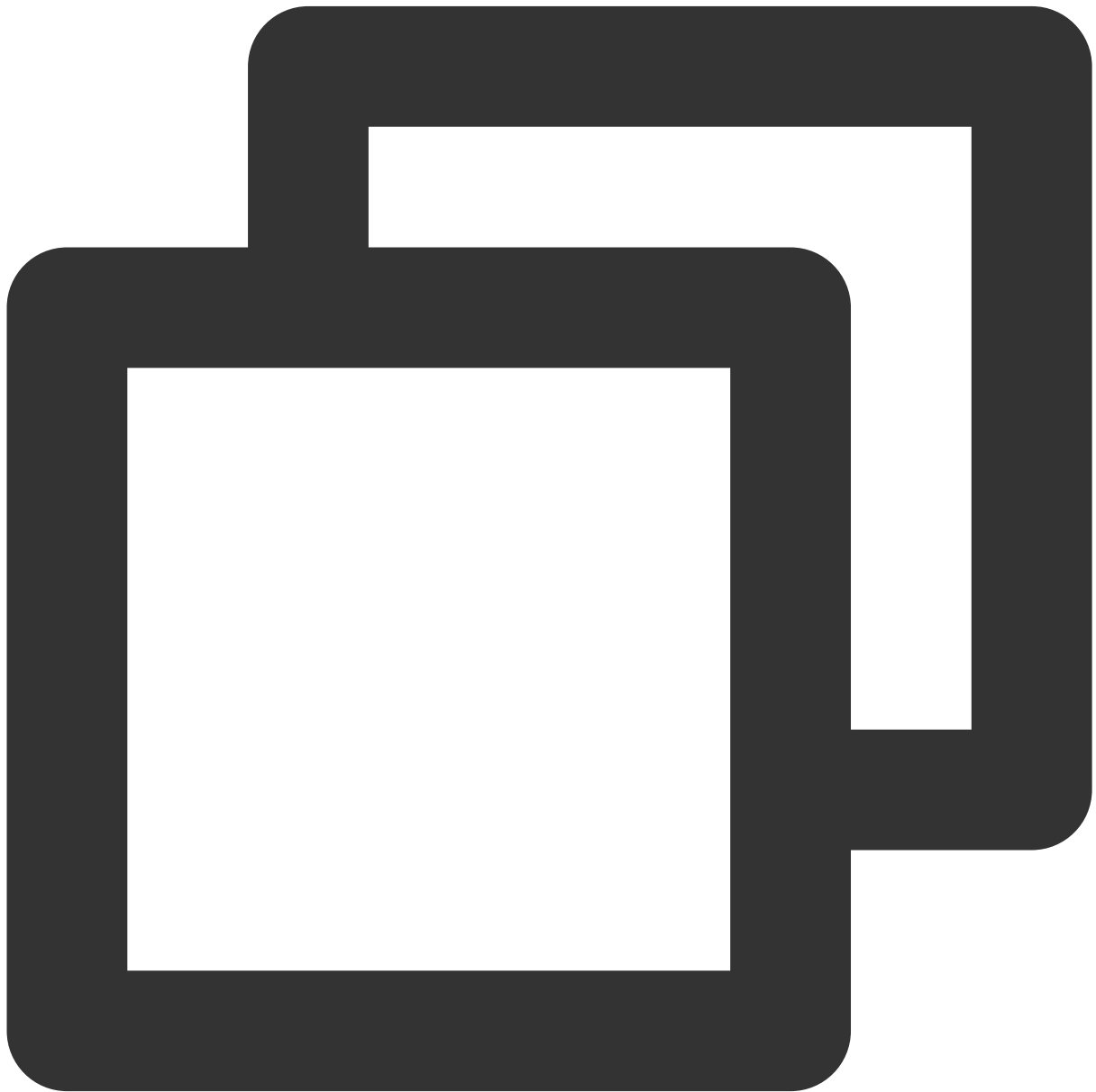
2. Run `DeserializeProtobufToRecordMsgObject` to deserialize the original message to a `RecordMsgObject` object.



```
// Customize a message deserializer to deserialize the original message to a `RecordMsgObject`  
@Override  
public RecordMsgObject deserialize(ConsumerRecord<byte[], byte[]> record) throws Exception {  
    RecordMsgObject obj = new RecordMsgObject();  
    obj.topic = record.topic();  
    obj.partition = record.partition();  
}
```

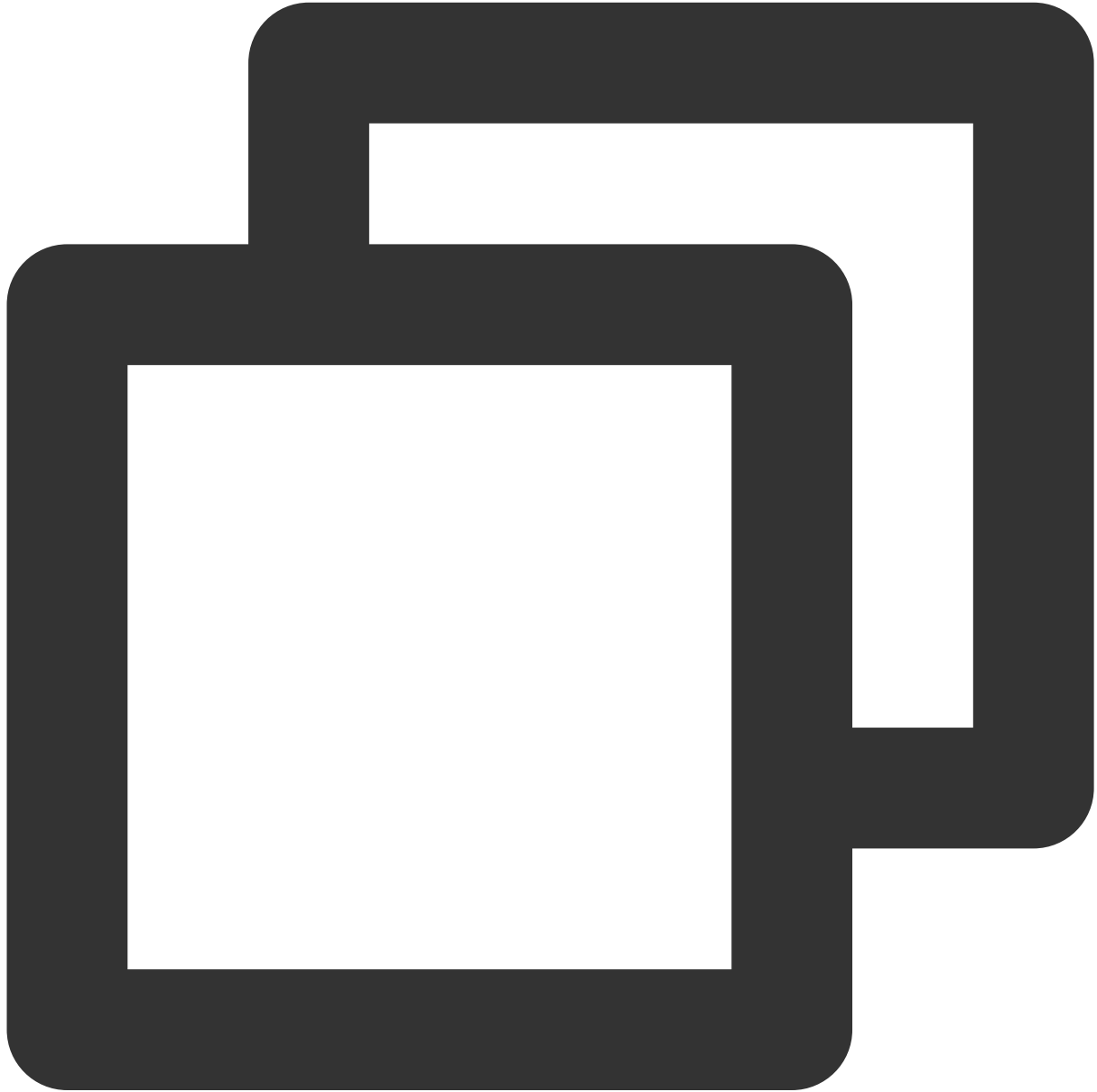
```
obj.offset = record.offset();  
obj.partitionSeq = getPartitionSeq(record);  
obj.key = new String(record.key());  
obj.headers = record.headers();  
// Here the binary value of `Envelope` will be received  
obj.value = record.value();  
return obj;  
}
```

3. Group the received messages by partition with the grouping logic implemented by `SubscribeMsgProcess`



```
//Put the received messages in different partitions
stream.keyBy(RecordMsgObject::getPartition)
    .process(new SubscribeMsgProcess(trans2sql)).setParallelism(1);
```

4. Use Protobuf to decode the binary sequence received in `SubscribeMsgProcess` into `Envelope` .

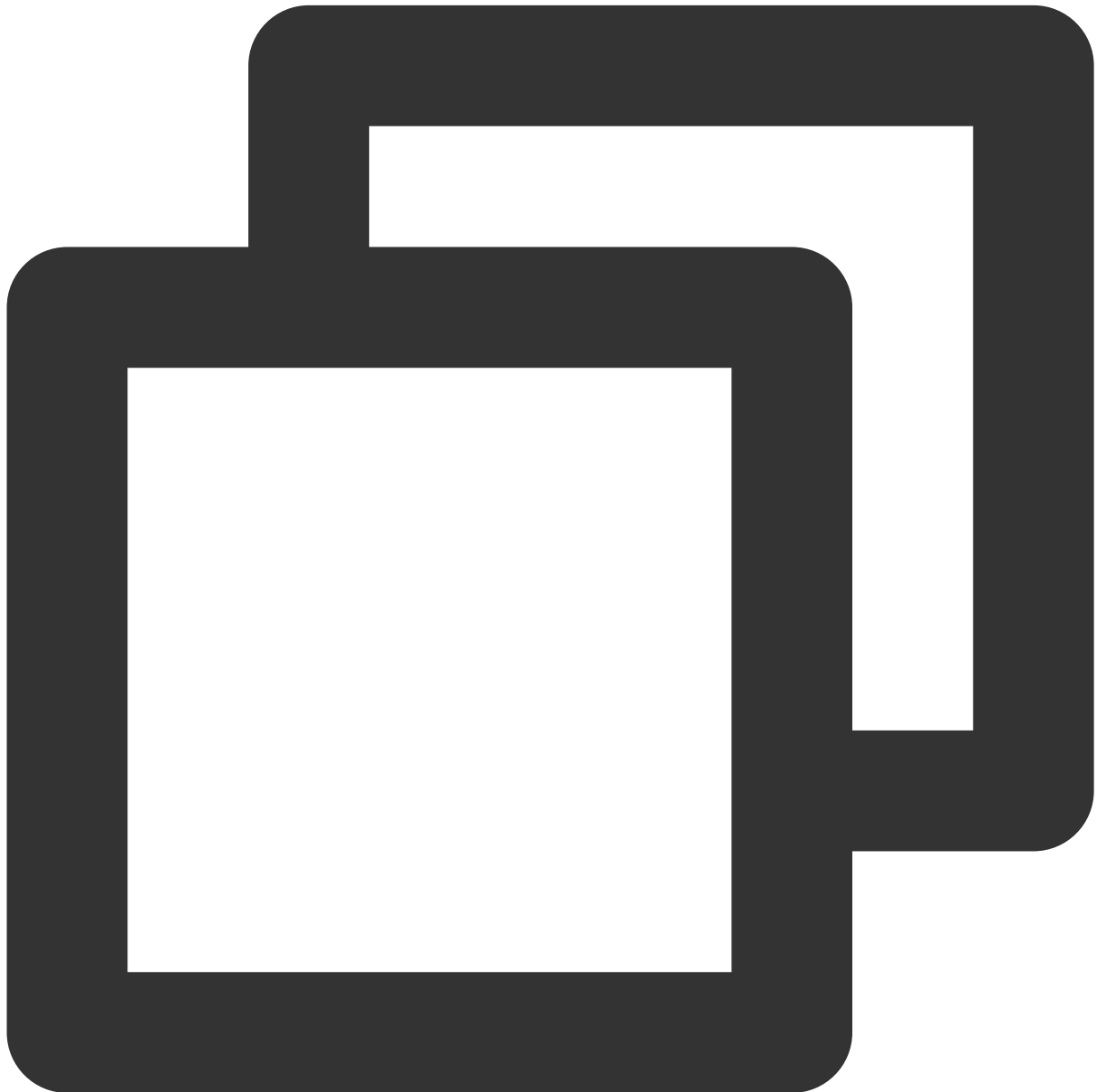


```
// Obtain an `Envelope` by deserialization. In this demo, only one `Envelope` can sto
SubscribeProtobufData.Envelope envelope = SubscribeProtobufData.Envelope.parseFrom(
if (1 != envelope.getVersion()) {
    throw new IllegalStateException(String.format("unsupported version: %d", envelo
}
```

**Note**

In this demo, the size of a single binlog event is not greater than that of a single Kafka message by default. When you use this demo for consumption, if the binlog size exceeds the Kafka message size, the split `Envelope` must be concatenated with the Flink's advanced feature "state processor API" so that the message body is complete. You need to handle this based on your business scenario. For more information, see [Flink Documentation](#).

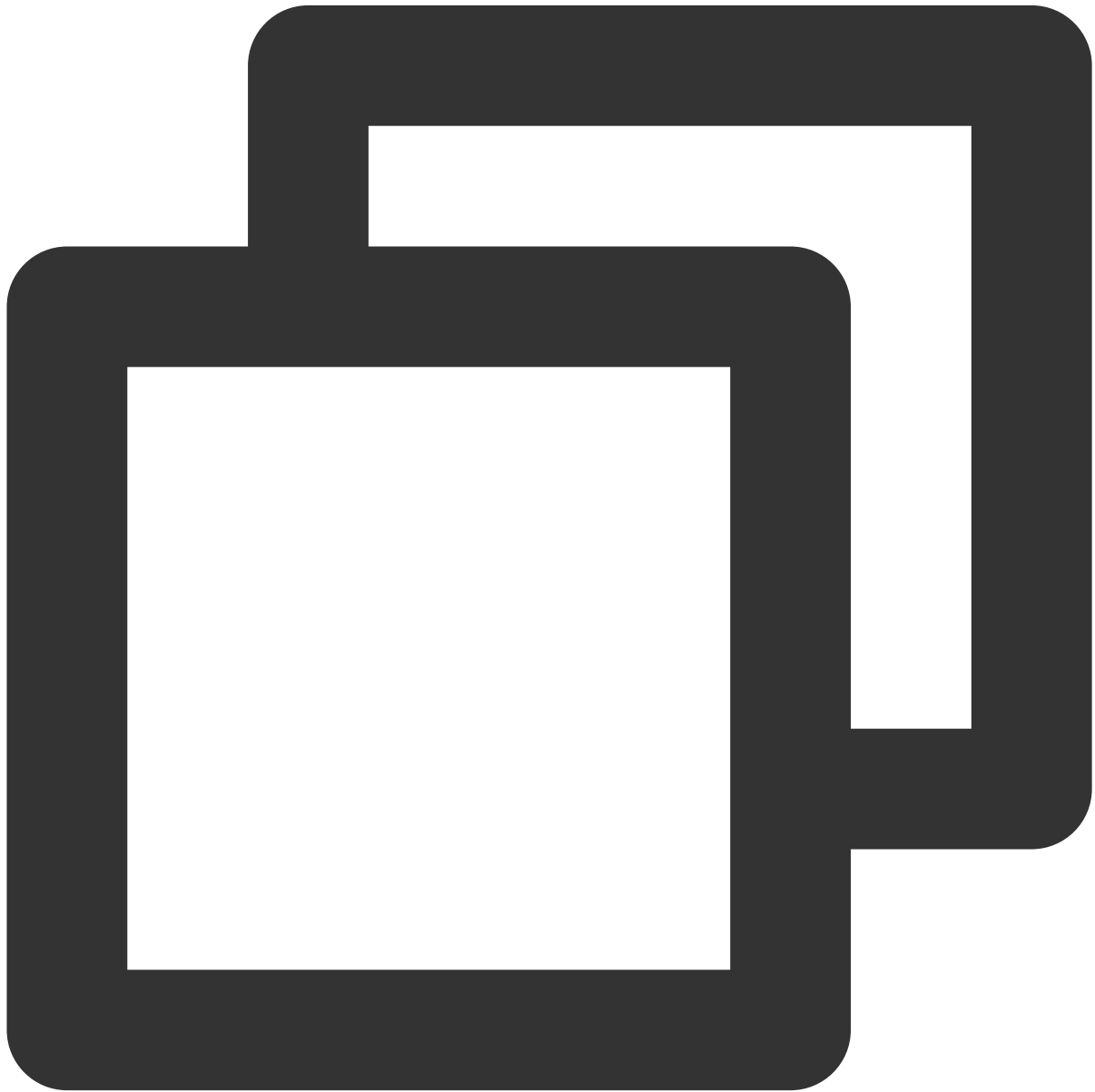
5. Use Protobuf to decode the binary sequence in the `data` field of the received `Envelope` into `Entries` .



```
// Deserialize `Entries`  
ByteString envelopeData = envelope.getData();
```

```
SubscribeProtobufData.Entries entries;  
if (1 == envelope.getTotal()) {  
    entries = SubscribeProtobufData.Entries.parseFrom(envelopeData.toByteArray());  
} else {  
    entries = SubscribeProtobufData.Entries.parseFrom(shardMsgMap.get(shardId).toBytes());  
    shardMsgMap.remove(shardId);  
}
```

6. Process `Entries.items` in sequence, and print the original `Entry` structure or convert it into a SQL statement.



```
// Traverse each `Entry` and print the SQL statement based on the type of `Entry`  
for (SubscribeProtobufData.Entry entry : entries.getItemsList()) {  
    onEntry(record.partition, record.offset, ps, entry, trans2sql);  
}
```

## Table API & Flink SQL

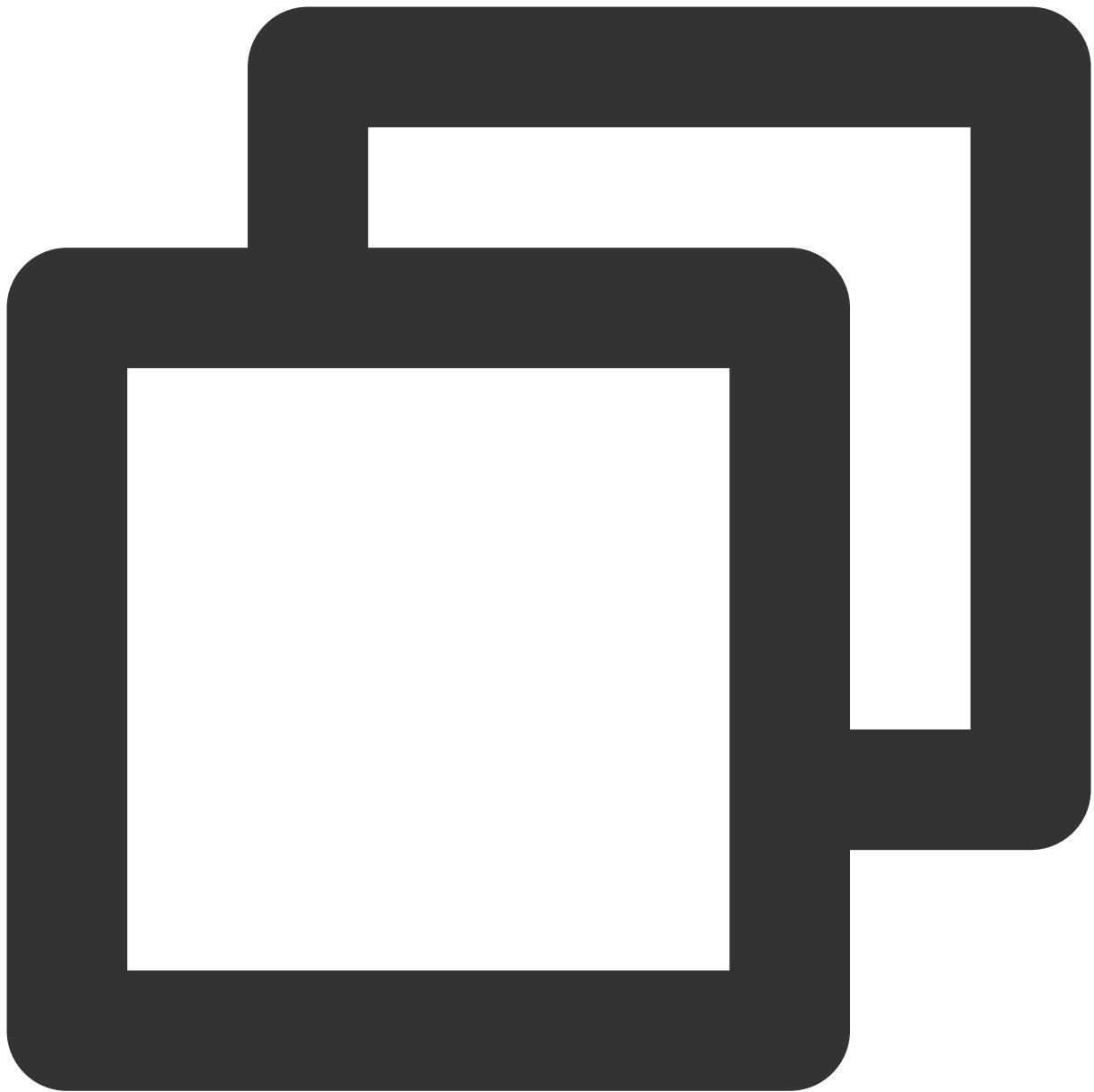
This demo only demonstrates the Flink client mode where DataStream API is used and doesn't apply to scenarios where Table API & Flink SQL is used. There are two ways to use the Table API & Flink SQL client mode

1. Convert DataStream into Table. For more information, see [DataStream API Integration](#).
2. Customize a connector based on Table API & Flink SQL. For more information, see [User-defined Sources & Sinks](#).

## Database Field Mapping and Storage

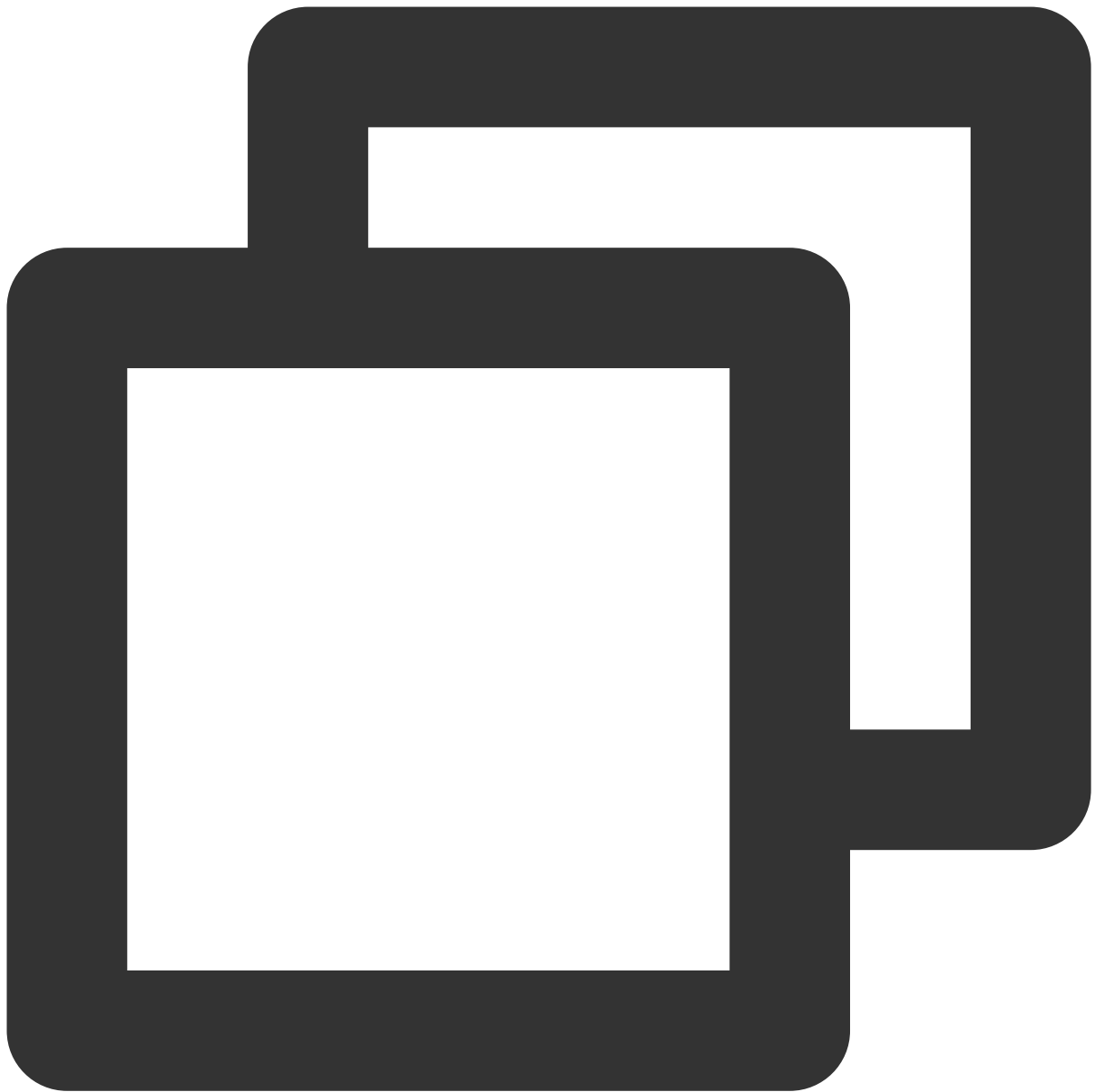
This section describes the mappings between database field types and data types defined in the Protobuf protocol.

A field value in the source database is structured as follows in the Protobuf protocol.



```
message Data {  
    DataType      dataType = 1;  
    string        charset  = 2; // The encoding (string) type of DataType_STRING,  
    string        sv       = 3; // The string value of DataType_INT8/16/32/64/UINT  
    bytes         bv       = 4; // The value of DataType_STRING/DataType_BYTES  
}
```

The field `DataType` refers to the type of stored fields. The values are as enumerated below:



```
enum DataType {  
    NIL      = 0; // The value is `NULL`  
    INT8     = 1;  
    INT16    = 2;  
    INT32    = 3;  
    INT64    = 4;  
    UINT8    = 5;  
    UINT16   = 6;  
    UINT32   = 7;  
    UINT64   = 8;  
    FLOAT32  = 9;
```



```

    FLOAT64 = 10;
    BYTES   = 11;
    DECIMAL = 12;
    STRING  = 13;
    NA      = 14; // The value does not exist (N/A).
}

```

The `bv` field stores the binary representation of `STRING` and `BYTES`; the `sv` field stores the string representation of `INT8/16/32/64/UINT8/16/32/64/DECIMAL`; the `charset` field stores the encoding type of `STRING`.

Mapping between the TDSQL for MySQL original type and `DataType` is as shown below (the

`MYSQL_TYPE_INT8/16/24/32/64` modified by `UNSIGNED` is respectively mapped to `UINT8/16/32/32/64`):

### Note

`DATE`, `TIME`, and `DATETIME` types don't support time zone.

The `TIMESTAMP` type supports time zone. Fields of this type will have their current time zone converted to Universal Time Coordinated (UTC) for storage, and vice versa for query.

The `MYSQL_TYPE_TIMESTAMP` and `MYSQL_TYPE_TIMESTAMP_NEW` fields carry the time zone information, which you can convert on your own when consuming data. For example, the format of the time data output by DTS is a string with time zone, such as `2021-05-17 07:22:42 +00:00`, where `+00:00` indicates the UTC time. You need to take into account the time zone information when parsing and converting the data.

TDSQL for MySQL Field Type	Protobuf DataType Value
<code>MYSQL_TYPE_NULL</code>	<code>NIL</code>
<code>MYSQL_TYPE_INT8</code>	<code>INT8</code>
<code>MYSQL_TYPE_INT16</code>	<code>INT16</code>
<code>MYSQL_TYPE_INT24</code>	<code>INT32</code>
<code>MYSQL_TYPE_INT32</code>	<code>INT32</code>
<code>MYSQL_TYPE_INT64</code>	<code>INT64</code>
<code>MYSQL_TYPE_BIT</code>	<code>INT64</code>
<code>MYSQL_TYPE_YEAR</code>	<code>INT64</code>
<code>MYSQL_TYPE_FLOAT</code>	<code>FLOAT32</code>
<code>MYSQL_TYPE_DOUBLE</code>	<code>FLOAT64</code>
<code>MYSQL_TYPE_VARCHAR</code>	<code>STRING</code>

MYSQL_TYPE_STRING	STRING
MYSQL_TYPE_VAR_STRING	STRING
MYSQL_TYPE_TIMESTAMP	STRING
MYSQL_TYPE_DATE	STRING
MYSQL_TYPE_TIME	STRING
MYSQL_TYPE_DATETIME	STRING
MYSQL_TYPE_TIMESTAMP_NEW	STRING
MYSQL_TYPE_DATE_NEW	STRING
MYSQL_TYPE_TIME_NEW	STRING
MYSQL_TYPE_DATETIME_NEW	STRING
MYSQL_TYPE_ENUM	STRING
MYSQL_TYPE_SET	STRING
MYSQL_TYPE_DECIMAL	DECIMAL
MYSQL_TYPE_DECIMAL_NEW	DECIMAL
MYSQL_TYPE_JSON	BYTES
MYSQL_TYPE_BLOB	BYTES
MYSQL_TYPE_TINY_BLOB	BYTES
MYSQL_TYPE_MEDIUM_BLOB	BYTES
MYSQL_TYPE_LONG_BLOB	BYTES
MYSQL_TYPE_GEOMETRY	BYTES

# Advanced Subscription Operations

## Setting Partitioning Policy

Last updated : 2024-07-08 16:52:02

### Overview

If you use multiple Kafka partitions, you can set a partitioning policy to route relevant business data to the same partition, making it easier to process the consumed data. DTS supports partitioning by table name, table name + primary key, or column and routes subscribed data to each Kafka partition based on hash rules.

**Kafka partitioning policy - By table name:** Partitions the subscribed data from the source database by table name. With this policy, data with the same table name is written to the same Kafka partition. During data consumption, data changes in the same table are always obtained sequentially.

**Kafka partitioning policy - By table name + primary key:** Partitions the subscribed data from the source database by table name and primary key. This policy is suitable for frequently accessed data. With this policy, frequently accessed data is distributed from tables to different partitions by table name and primary key, so as to improve the concurrent consumption efficiency.

**Custom partitioning policy:** Database and table names of the subscribed data are matched through a regex first. Then, matched data is partitioned by table name, table name + primary key, or column.

1 Select Instance

2 Subscription Type and Object

3 Pre-verification

Subscription ID / Name

subs- (name- )

MySQL Instance

cdb- (dts- )

Subscription Type

☒ Data Update ☒ Structure Update ☒ Full

Structure update includes the structure creation, deletion, and modification of all objects subscribed to the entire instance.

Format of Subscribed Data

☒ ProtoBuf ☐ Avro ☐ JSON

ProtoBuf and Avro are more efficient binary formats, while JSON is an easier-to-use lightweight text format.

Kafka Partitioning Policy

☒ By table name ☐ By table name + primary key

Custom Partitioning Policy

☒

Custom Partitioning Policy

Custom partitioning rules are applied to objects that meet the following database/table rules. These rules must be described according to the RE2 syntax. For details, see [Syntax](#).

**Partitioning policy description:**

By table name: Data tables will be matched if their database and table names are described in the regexes below. Messages with the same table name are always put in the same partition, and data changes in the same table are always obtained sequentially.

By table name + primary key: Data tables will be matched if their database and table names are described in the regexes below. Data is partitioned by the combination of table name and primary key, and data in the same table is partitioned by primary key.

By column: Data tables will be matched if their database and table names are described in the regexes below. Data (including data in the same table) is partitioned by column name.

Database Name Match	Table Name Match	Partitioning Policy	Custom Partition Column	Operation
<input type="text" value="^AS"/>	<input type="text" value="test\$"/>	By column	<input type="text" value="class"/>	<input checked="" type="checkbox"/> Delete

Add

Strategy Combo Result

When you enable the custom partitioning policy option, your custom partitioning policies will be applied first, followed by the Kafka partitioning policies.

The data matched by the regexes `^AS` and `test$` will be partitioned by column `class` and then routed to Kafka partitions.

The data in a table that cannot be partitioned using the custom partitioning policies will be routed to Kafka partitions by default policy (By table name).

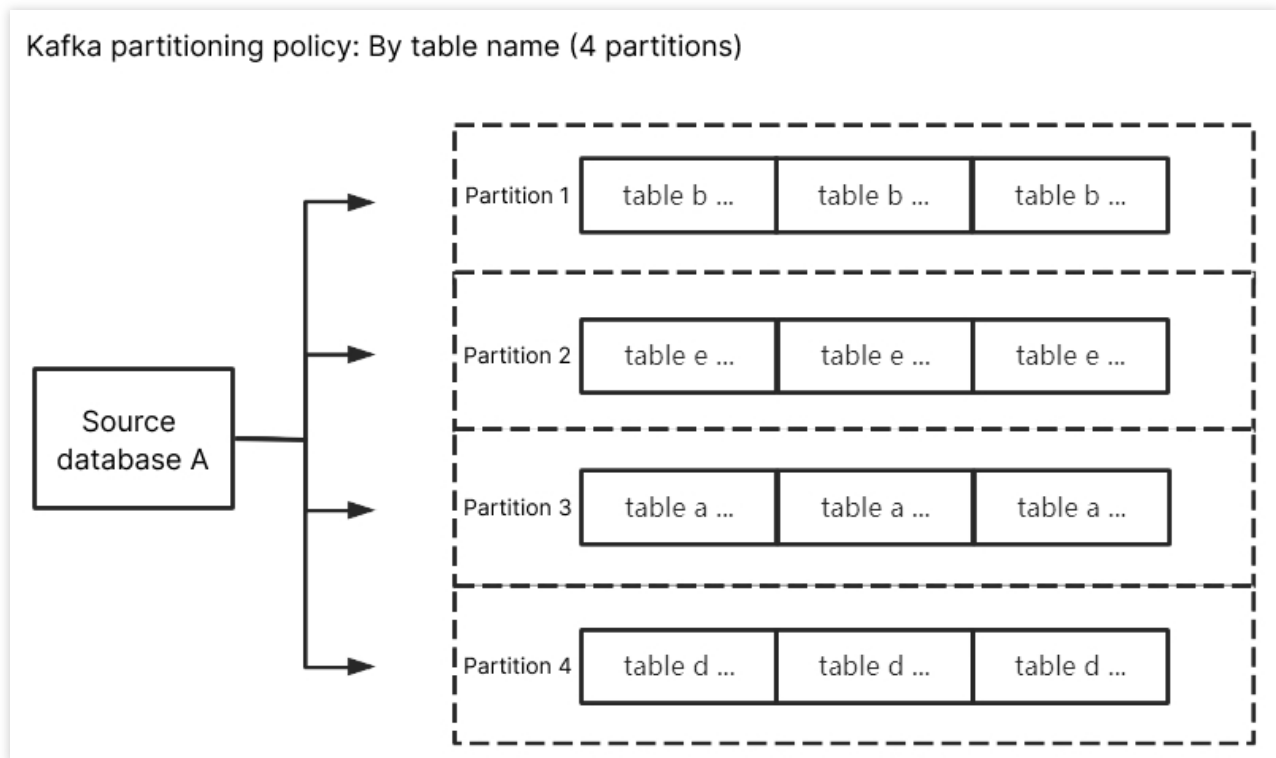
Previous

Save

# Kafka Partitioning Policy

## By table name

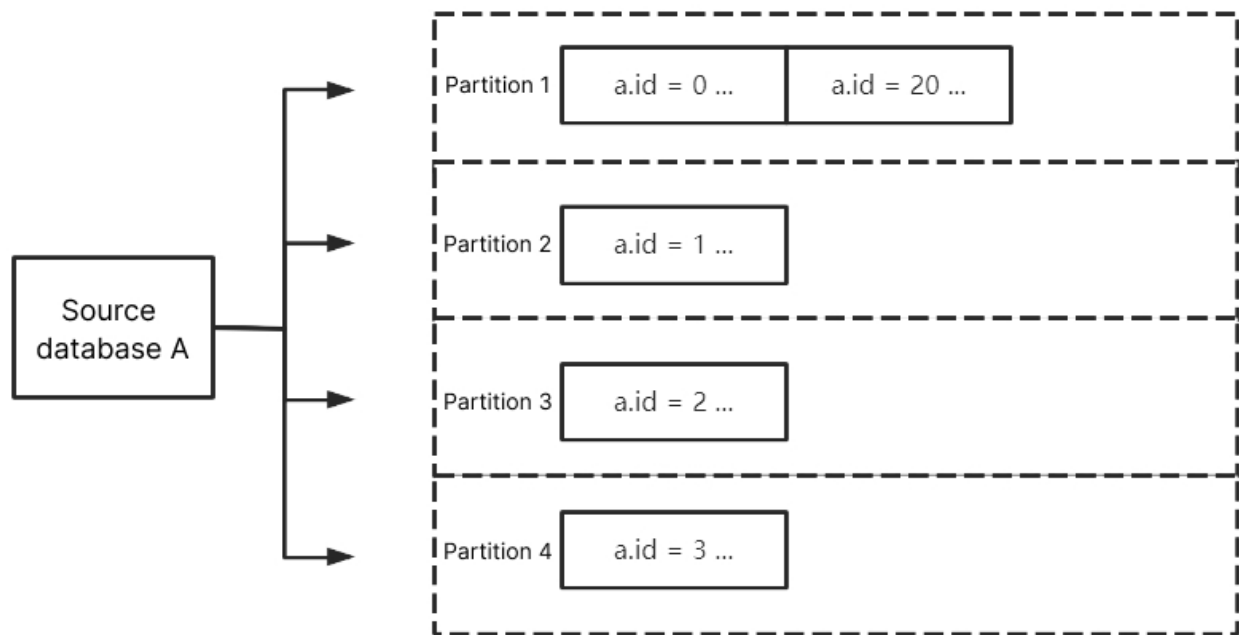
When the source database has a data change, the changed data will be written to the subscription topic. After **By table name** is selected, subscribed data with the same table name will be written to the same Kafka partition. During data consumption, it can be guaranteed that data changes in the same table are always obtained sequentially.



## By table name + primary key

If **By table name** is selected, data from a frequently accessed table is written to the same Kafka partition, which brings a huge pressure to the partition. You can distribute such data to different partitions by selecting **By table name + primary key**, so as to improve the concurrent consumption efficiency.

Kafka partitioning policy: By table name + primary key (4 partitions)



## Custom Partitioning Policy

With a custom partitioning policy, database and table names are matched through a regex first. Then, matched data is partitioned by table name, table name + primary key, or column.

### Match rules

Database, table, and table name match rules support RE2 regex. For the specific syntax, see [Syntax](#).

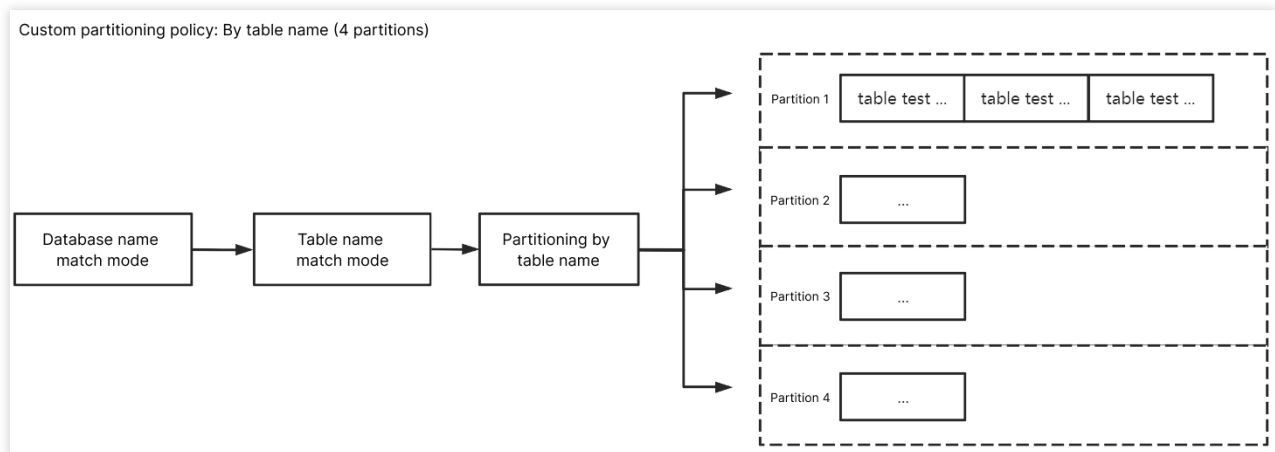
Database/Table name match rules match **database/table names** based on regex. To exactly match a database or table name, you need to add start and end symbols, such as `^test$` for the `test` table.

Column name match rules match columns through `==` in a case-insensitive way.

The custom partitioning policy is used for data match first. If it misses, data will be matched based on the configured Kafka partitioning policy. If there are multiple custom partitioning policies, they will be matched one by one from top to bottom.

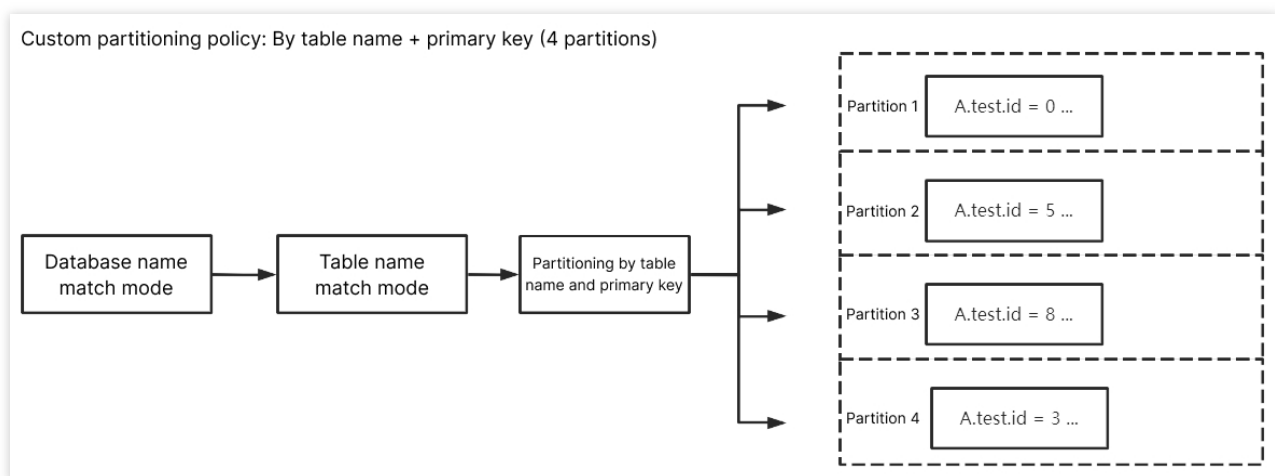
### By table name

Enter `^A$` for **Database Name Match** and `^test$` for **Table Name Match**. After **By table name** is selected, the subscribed data of table `test` in database `A` will be written to the same partition. Data of other unmatched tables will be written according to the policy set in **Kafka Partitioning Policy**.



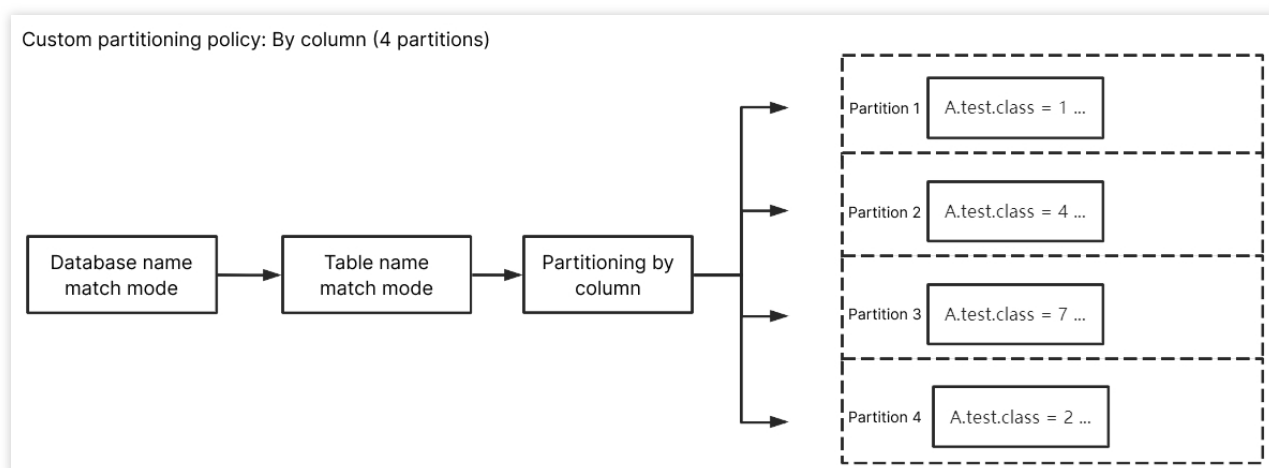
### By table name + primary key

Enter `^A$` for **Database Name Match** and `^test$` for **Table Name Match**. After **By table name + primary key** is selected, the subscribed data of table `test` in database `A` will be randomly written to different partitions by primary key, and data records with the same primary key will be written to the same partition eventually. Data of other unmatched tables will be written according to the policy set in **Kafka Partitioning Policy**.



### By column

Enter `^A$` for **Database Name Match**, `^test$` for **Table Name Match**, and `class` for **Custom Partition Column**. After **By column** is selected, the subscribed data of column `class` of table `test` in database `A` will be randomly written to different partitions. Data of other unmatched columns/tables will be written according to the policy set in **Kafka Partitioning Policy**.



# Fix for Verification Failure

## Database Connection Check

Last updated : 2024-07-08 16:54:20

### Check Details

The source and target databases need to be normally connected, and if not, the error message "Failed to connect to the source database" will be displayed.

### Causes

The network or server where the source database resides has a security group or firewall configured.

The source IP addresses are blocked in the source database.

The network port is closed.

The database account or password is incorrect.

## Security Group or Firewall Configured in Network or Server of Source Database

### Check method

A security group is similar to a firewall. It is a group of network security settings for databases in the cloud.

Check as follows based on the actual conditions:

Check whether the server where the source database resides is configured with firewall policies.

Windows: open Control Panel and find the Windows Defender Firewall and check whether firewall policies are configured.

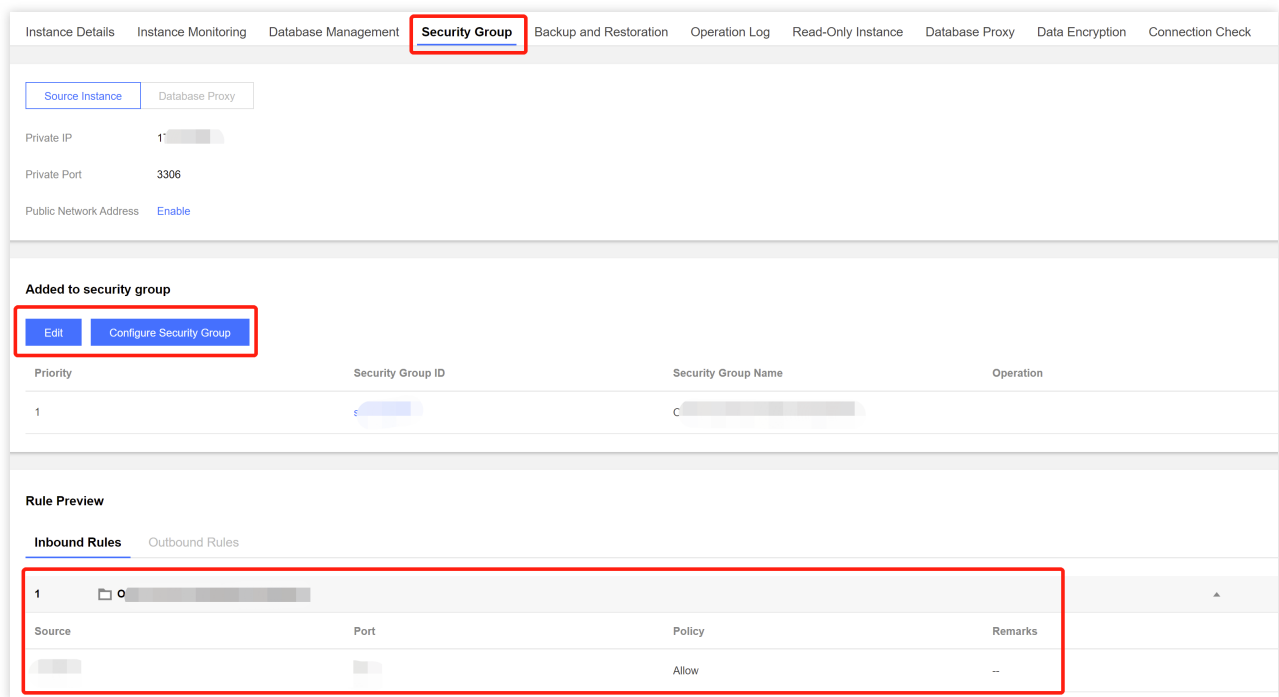
Linux: run the `iptables -L` command to check whether the server is configured with firewall policies.

Check whether DTS IP range is blocked in the security group of the database.

1.1 Log in to the [corresponding database](#) and click an instance ID in the instance list to enter the instance management page.

1.2 On the instance management page, select the **Security Group** tab and check whether there are policies blocking the SNAT IP range of DTS.





## Fix

Fix it as follows based on the actual conditions:

The firewall is enabled on the server:

1.1 Disable the server firewall, log in to DTS, and run the verification task again.

### Note:

This method is applicable to both Windows and Linux.

1.2 Set the DTS IP range policy to **Allow**.

The SNAT IP range of DTS is blocked in the security group:

1.1 Click the corresponding security group ID on the **Security Group** tab.

2. Set the DTS IP range policy to **Allow**.

## Source IP Addresses Blocked in Source Database

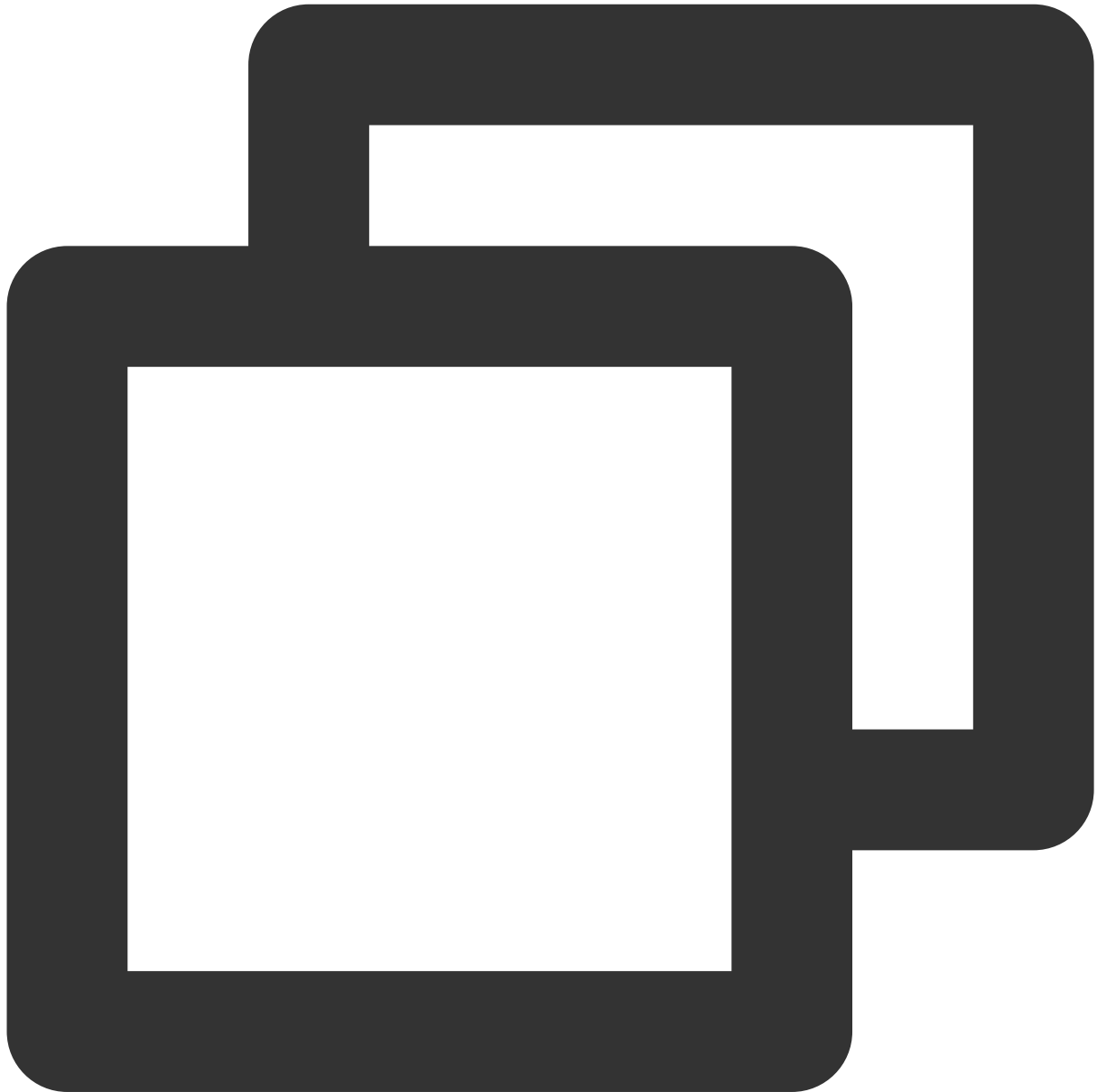
### Check method

#### MySQL

On the server where the source database is deployed, use the database account and password entered in the data migration task to connect to the source database. If the database can be normally connected, the source IP address may be blocked in the source database.

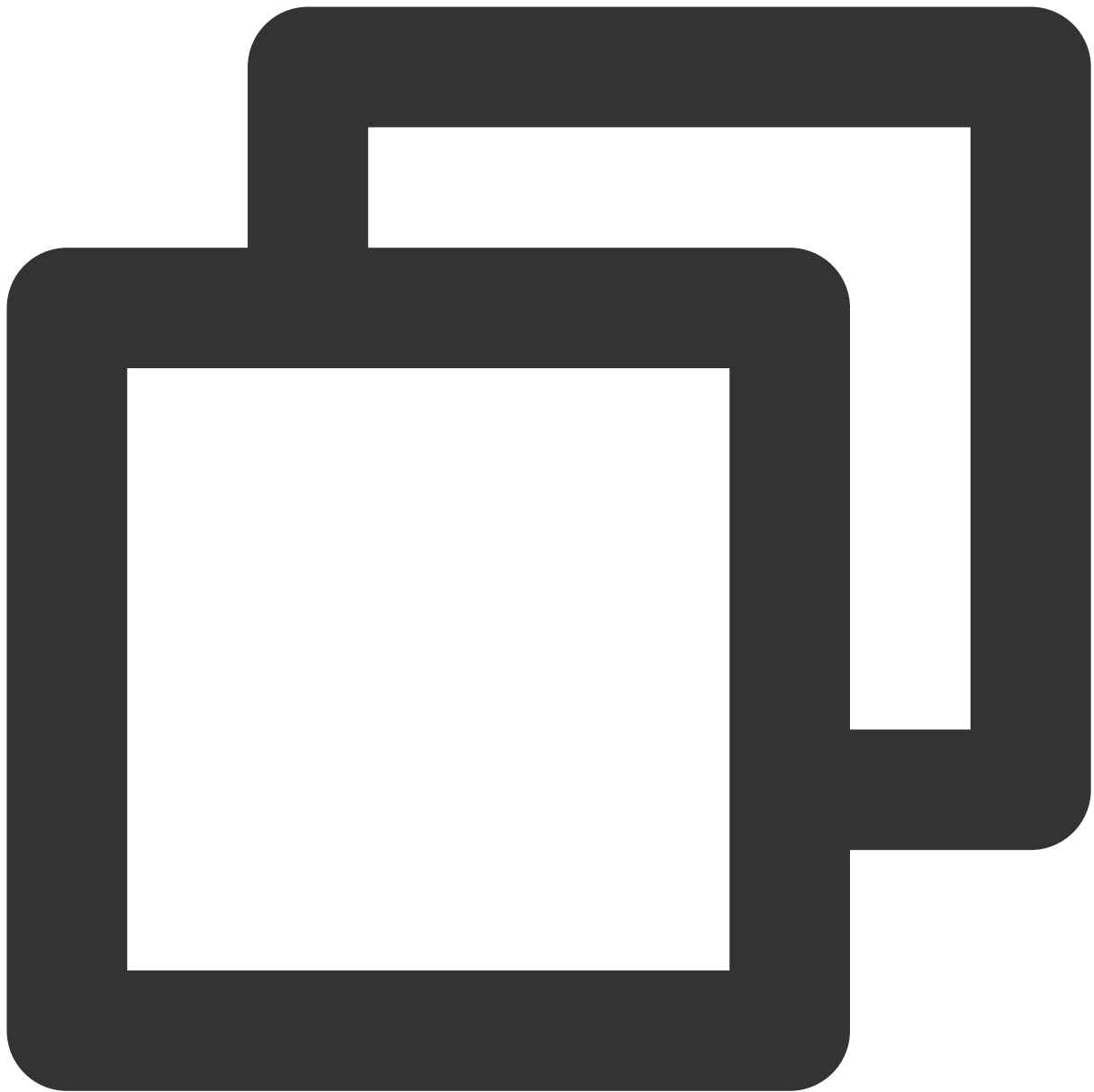
For self-built database, you need to check the `bind-address` configuration in the database. If it is not `0.0.0.0`, the IP is blocked.

If the source database is MySQL, you can use the MySQL client to connect to it, run the following SQL statement, and check whether the list of authorized IP addresses contains the SNAT IP addresses of DTS in the output result. When granting database permissions to users, the authorized IPs must include the SNAT IPs; otherwise, they may be blocked; for example:



```
root@10.0.0.0/8 // Authorize users to access through `10.0.0.0/8`, and other IPs w
root@%          // Authorize users to access all IPs, which should include the SNA
```

You can verify as follows:



```
select host,user,authentication_string,password_expired,account_locked from
mysql.user WHERE user='[\\$Username]';    // `[\\$Username]` is the database account
```

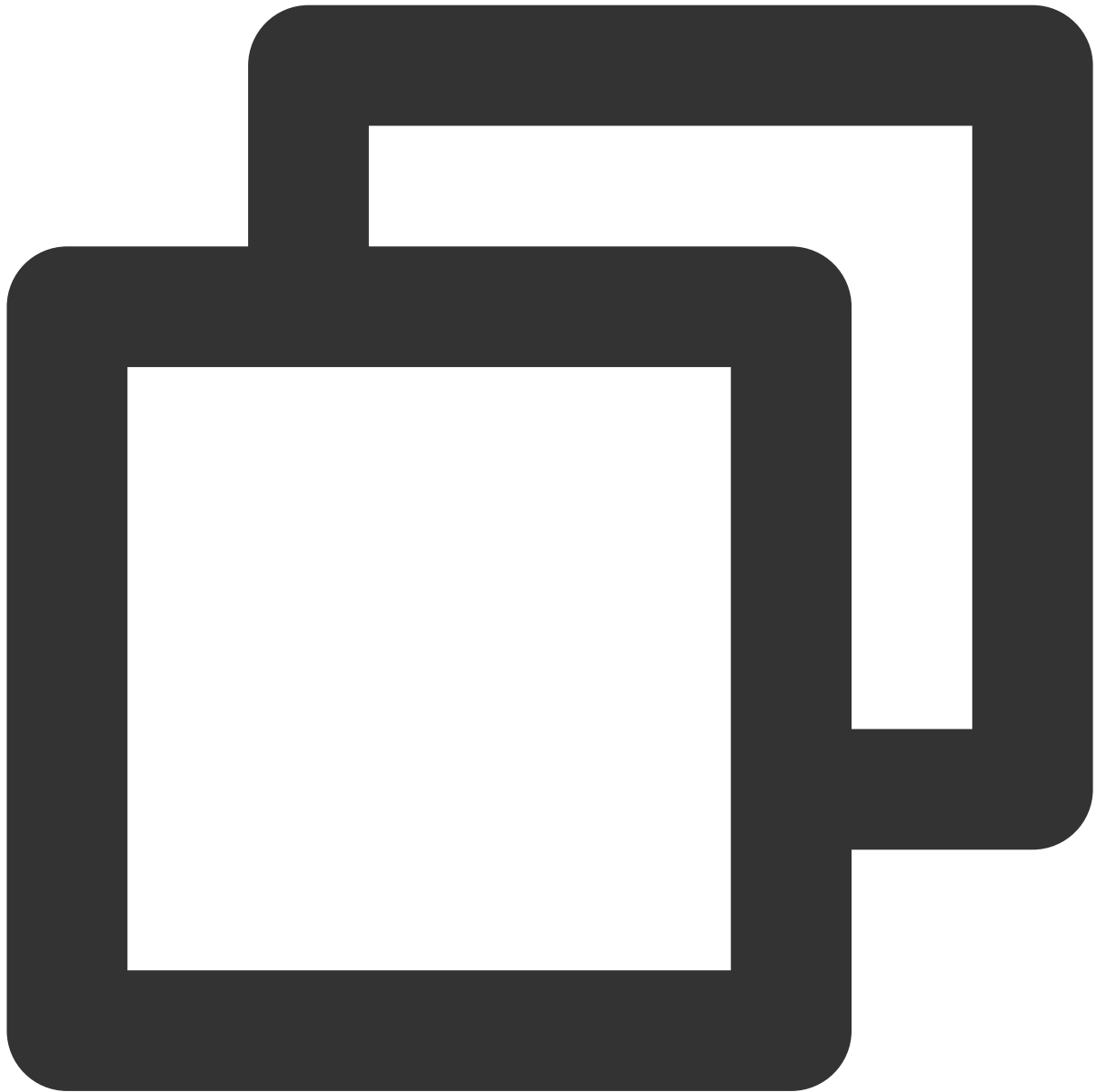
### SQL Server

Check whether there is an endpoint or trigger that blocks the access source IP address in the source database.

### PostgreSQL

If the source database is another database in the cloud, check whether the secure access policies in the source database have restrictions. Check as follows according to the specific cloud vendor:

If the source database is a self-built PostgreSQL database, enter the `data` directory in the `$PGDATA` directory, find the `pg_hba.conf` file, and check whether the file contains a `deny` policy or only allows access from certain IP addresses over the network.



```
# cat pg_hba.conf
local    replication    all                                     trust
host     replication    all             127.x.x.1/32          trust
host     replication    all             ::1/128             trust
host     all             all             0.0.0.0/0           md5
host     all             all             172.x.x.0/20        md5
```

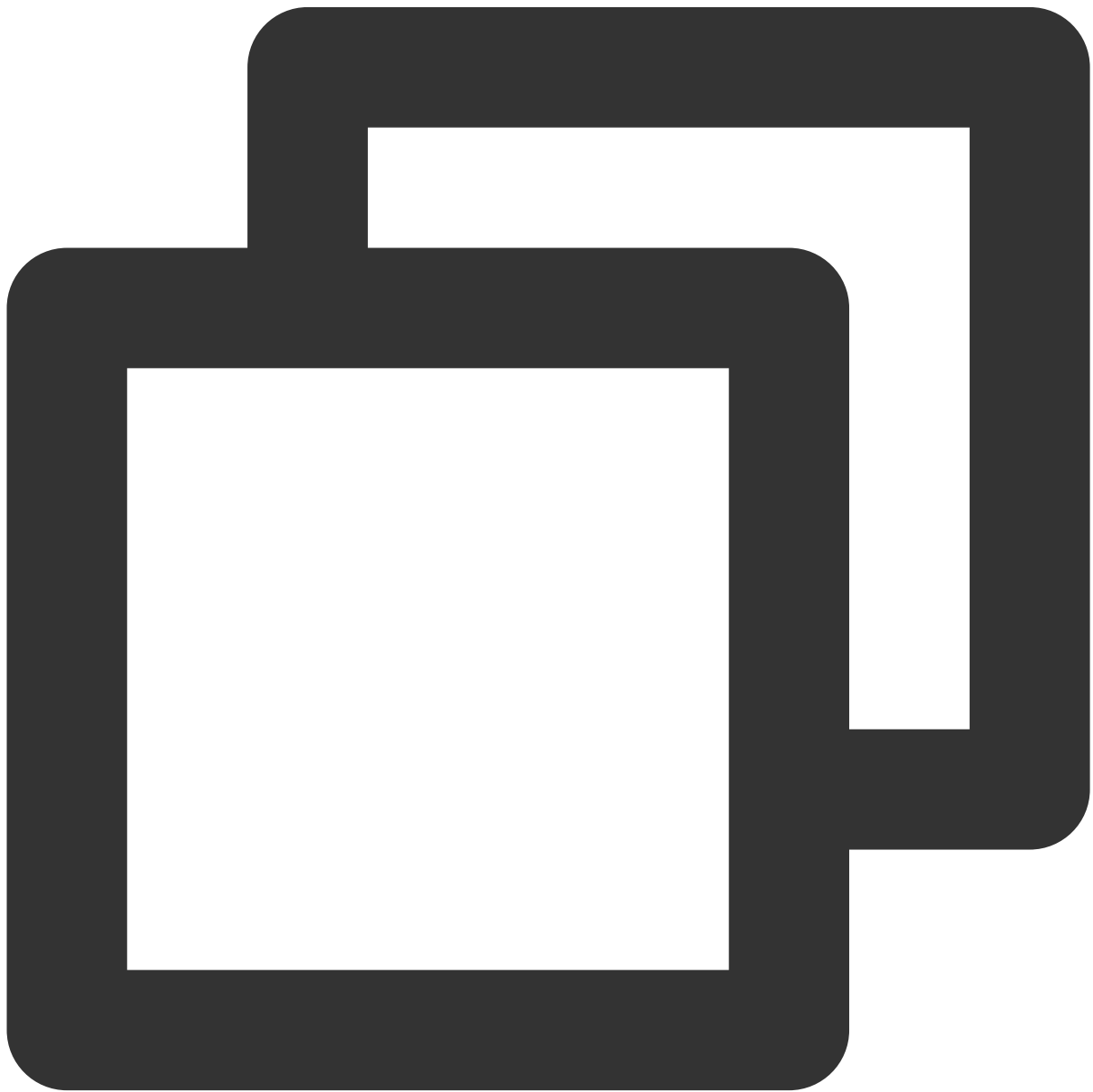
## MongoDB

For self-built database, you need to check the `bind-address` configuration in the database. If it is not `0.0.0.0`, the IP is blocked.

## Fix

## MySQL

1. If the source database is MySQL, run the following SQL statement in it to authorize the user configured in the data migration task.



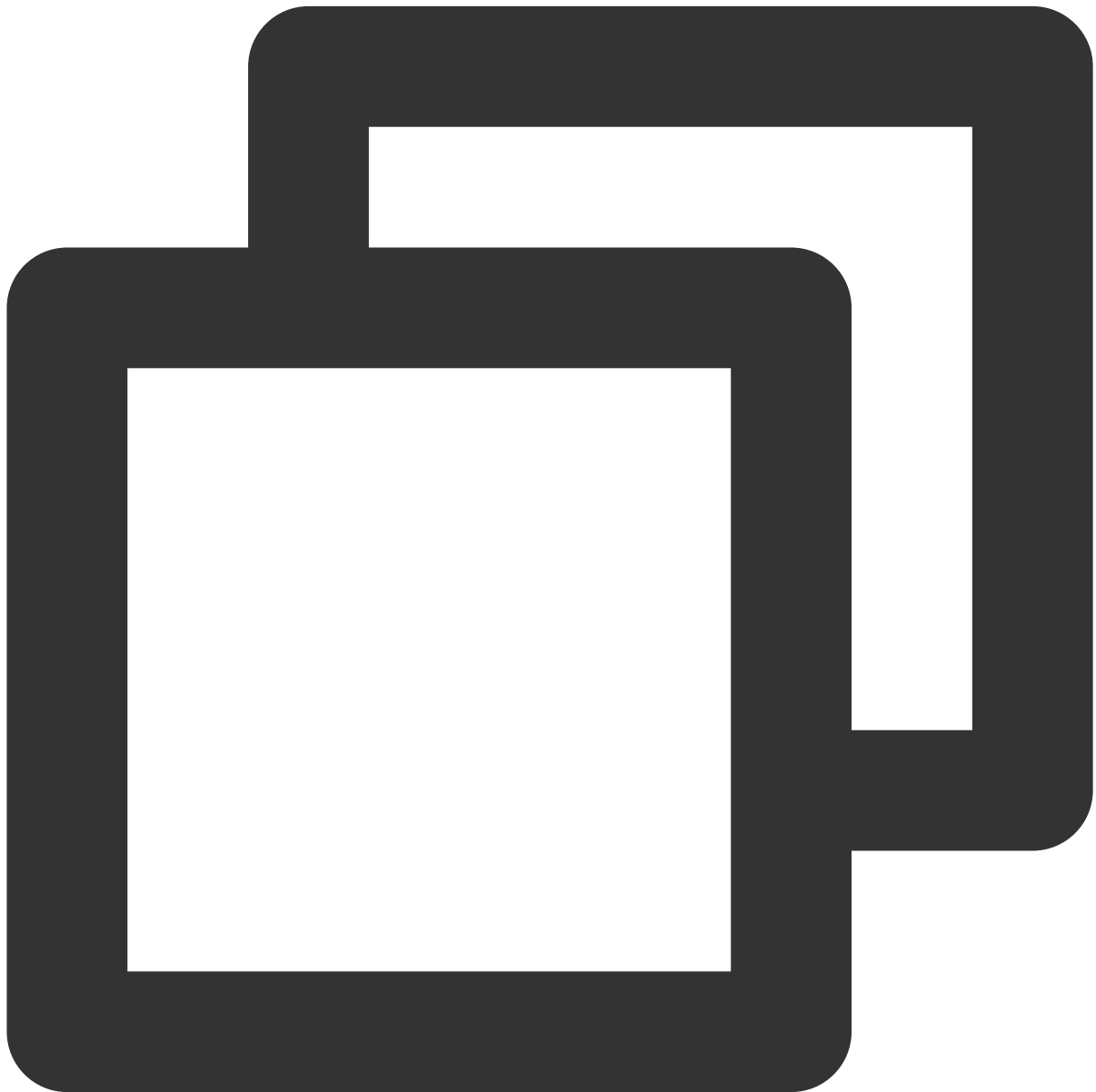
```
mysql> grant all privileges on . to '[\\$UserName]@'%; // `[\\$Username]` is the
mysql> flush privileges;
```

2. For a self-built database, if the `bind-address` configuration is incorrect, modify it as instructed below.

2.1. Add the following content to the `/etc/my.cnf` file:

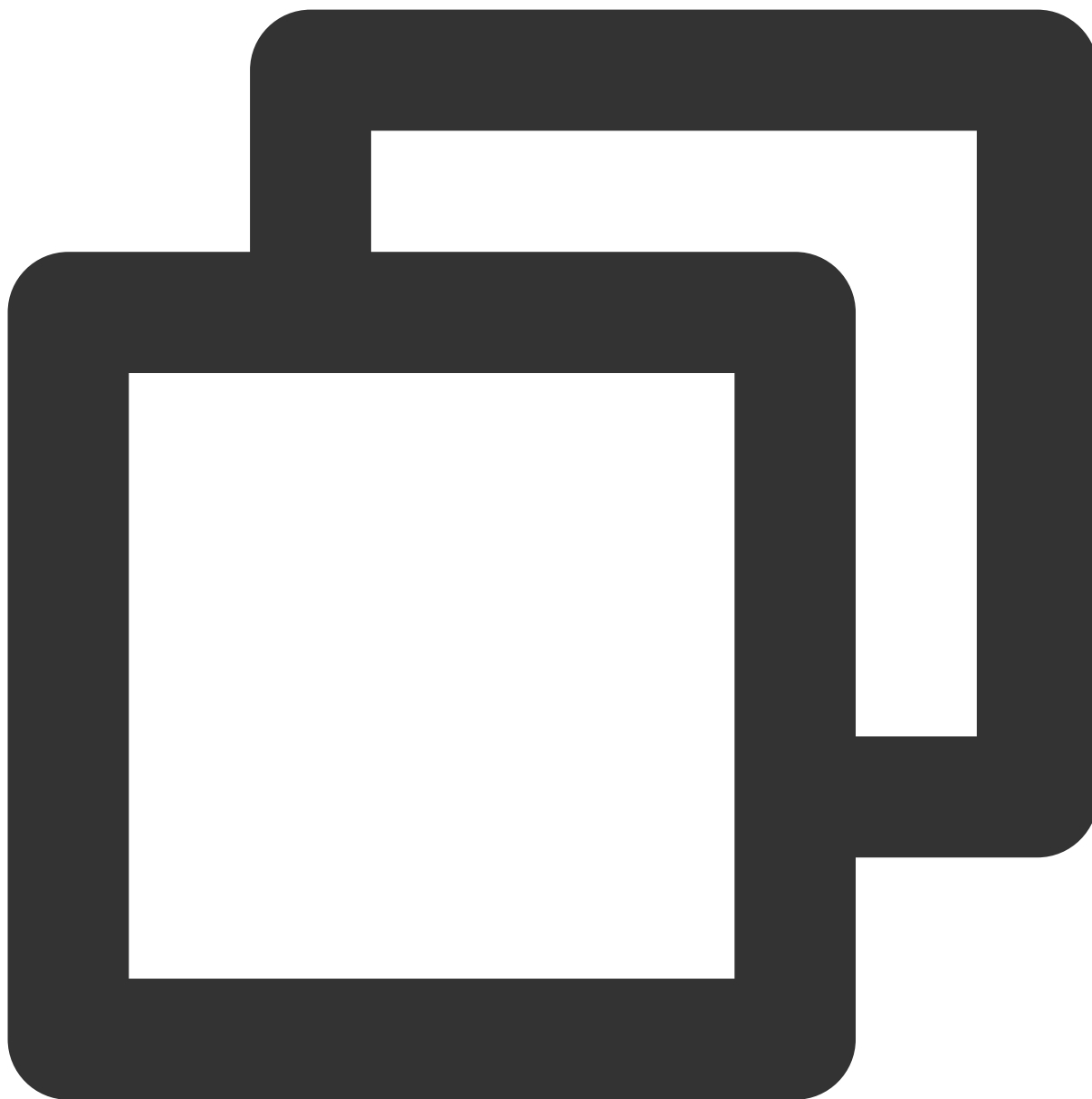
**Note:**

The default path of the `my.cnf` configuration file is `/etc/my.cnf`, subject to the actual conditions.



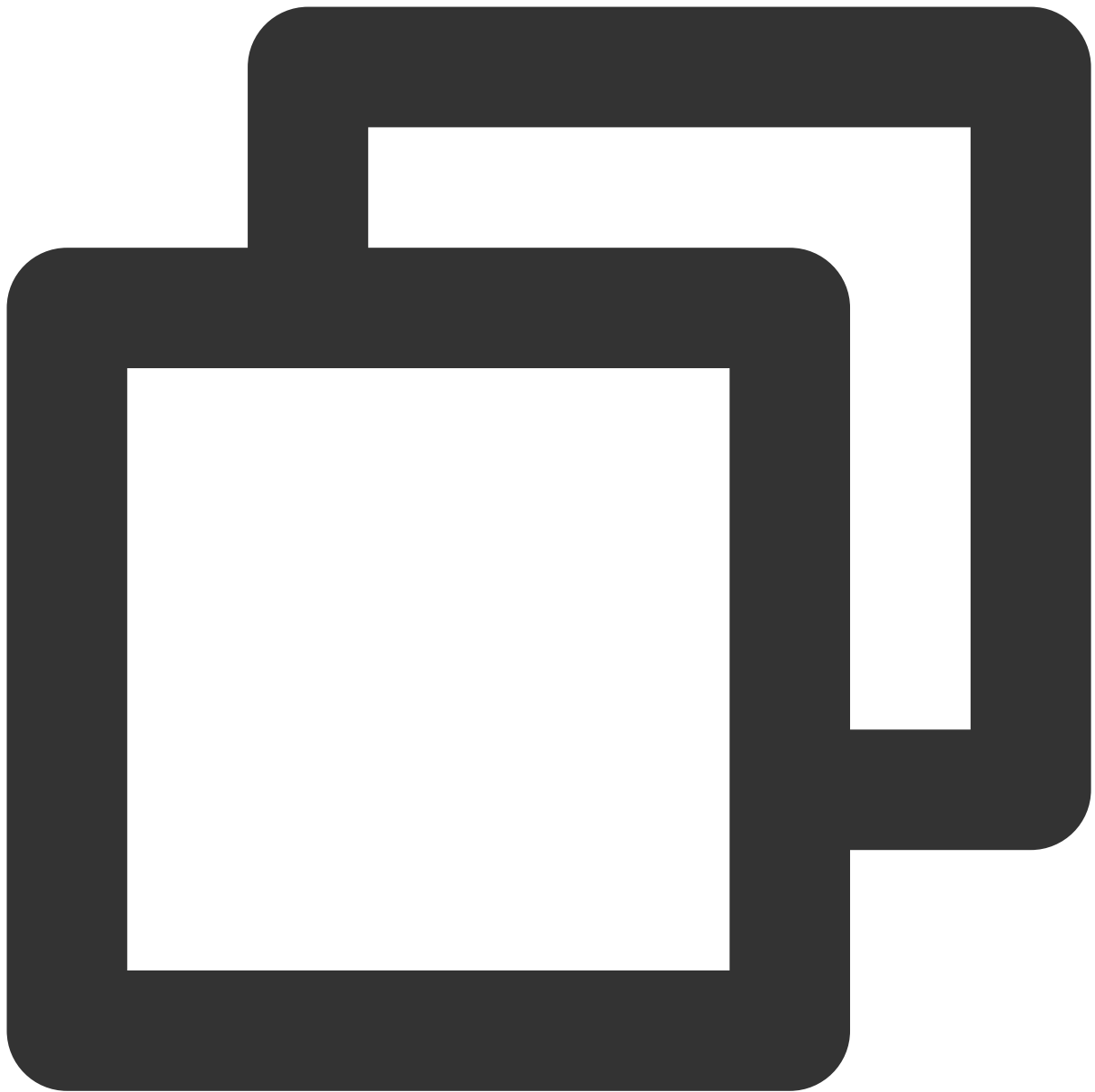
```
bind-address=0.0.0.0 # All IP addresses or specified addresses
```

## 2.2. Restart the database.



```
service mysqld restart
```

## 2.3. Check whether the configuration takes effect.



```
netstat -tln
```

3. Run the verification task again.

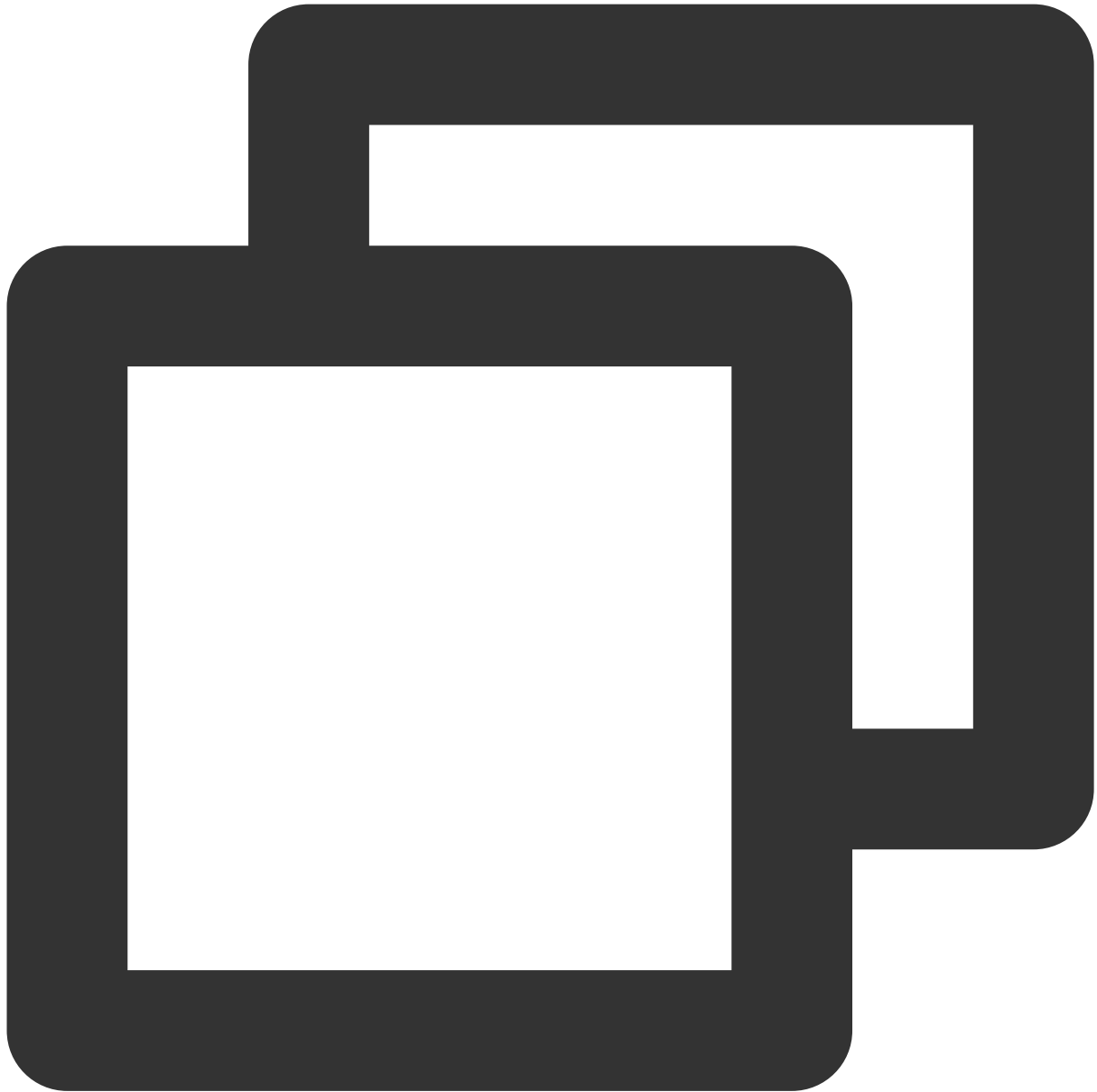
### SQL Server

Disable the firewall or trigger.

### PostgreSQL

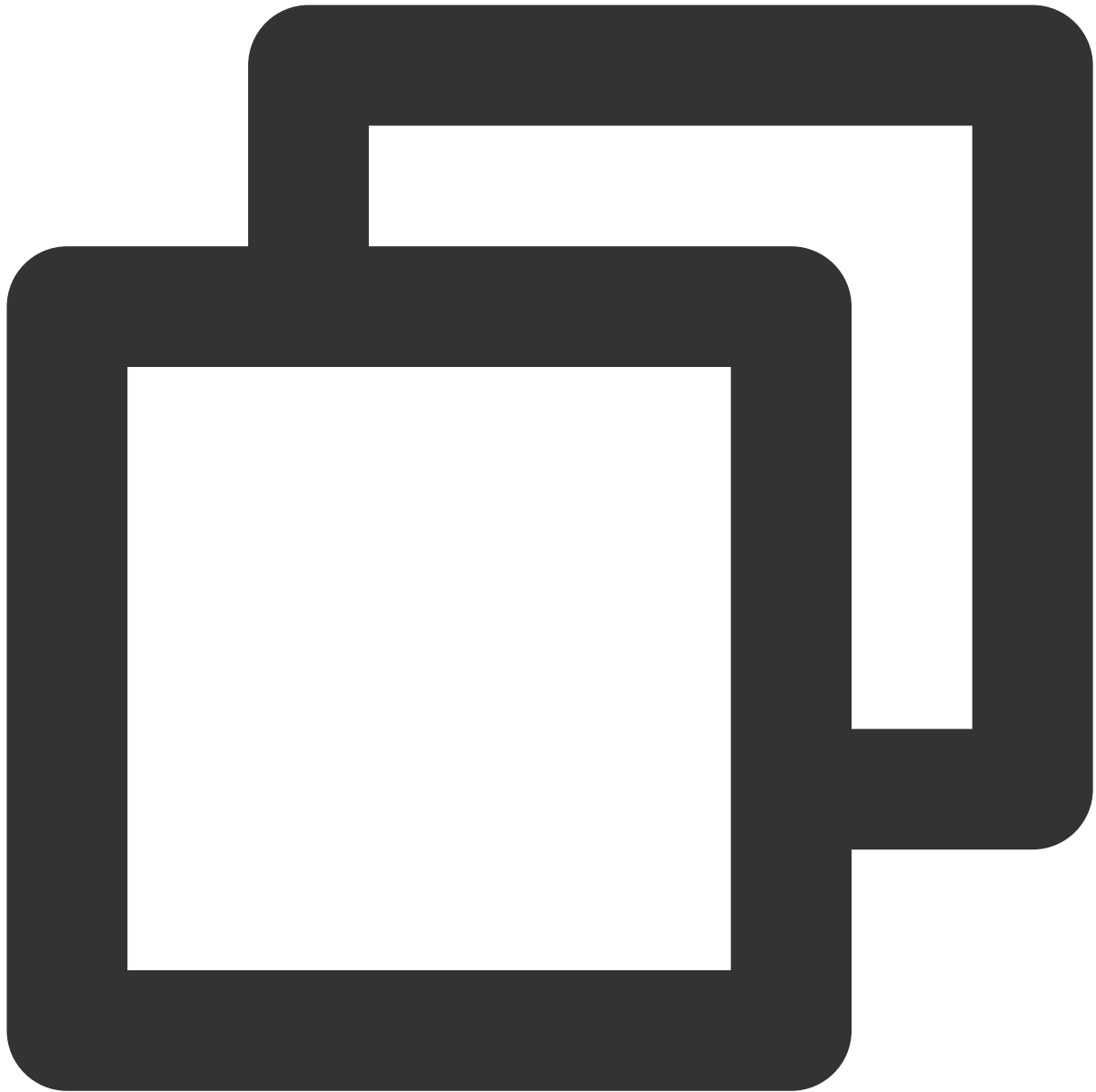


1. Add an access policy allowing the DTS IP range to the `pg_hba.conf` file or temporarily open all IP ranges in the access policy during migration. For example, add the following line to the `pg_hba.conf` file:



```
host      all             all             0.0.0.0/0      md5
```

2. After the modification is completed, you can restart the database to make the configuration take effect:



```
pg_ctl -D $PGDATA restart
```

3. Run the verification task again.

### MongoDB

Configure `bind-address` as instructed in [MySQL](#).

## Closed Network Port

## Check method

Below are the default ports for common databases. You need to check whether they are opened, and if not, open them based on the actual conditions:

MySQL: 3306

SQL Server: 1433

PostgreSQL: 5432

MongoDB: 27017

Redis: 6379

## Fix

Open the corresponding database port.

If the source database is SQL Server, you need to open the file sharing service port 445 at the same time.

# Incorrect Database Account or Password

## Check method

Log in to the source database to check whether the account and password are correct.

## Fix

Modify the data migration task in the [DTS console](#), enter the correct database account and password, and run the verification task again.

# Peripheral Check

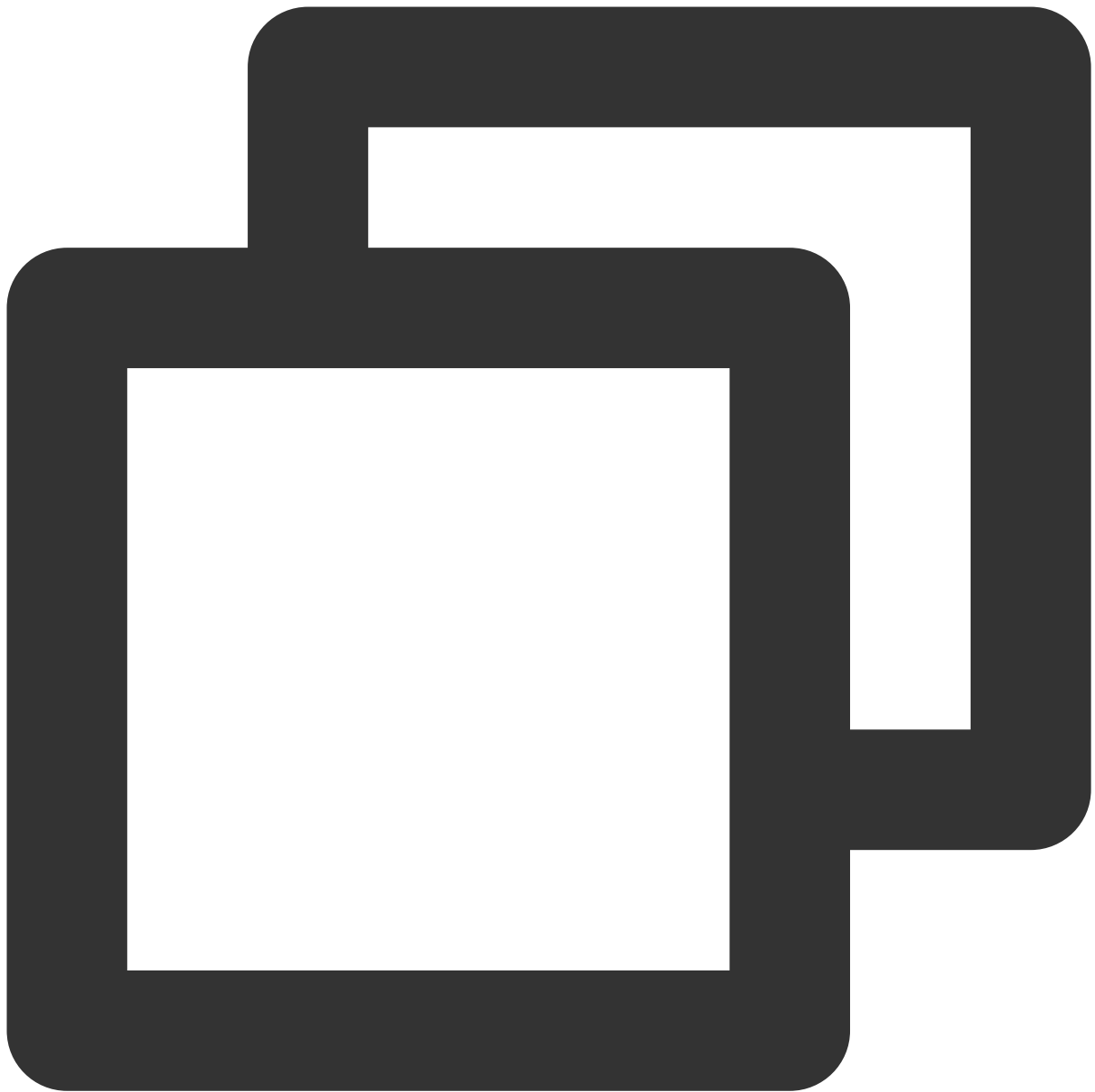
Last updated : 2024-08-01 14:47:33

## Check Details

The `innodb_stats_on_metadata` environment variable in the source database must be set to `OFF` .

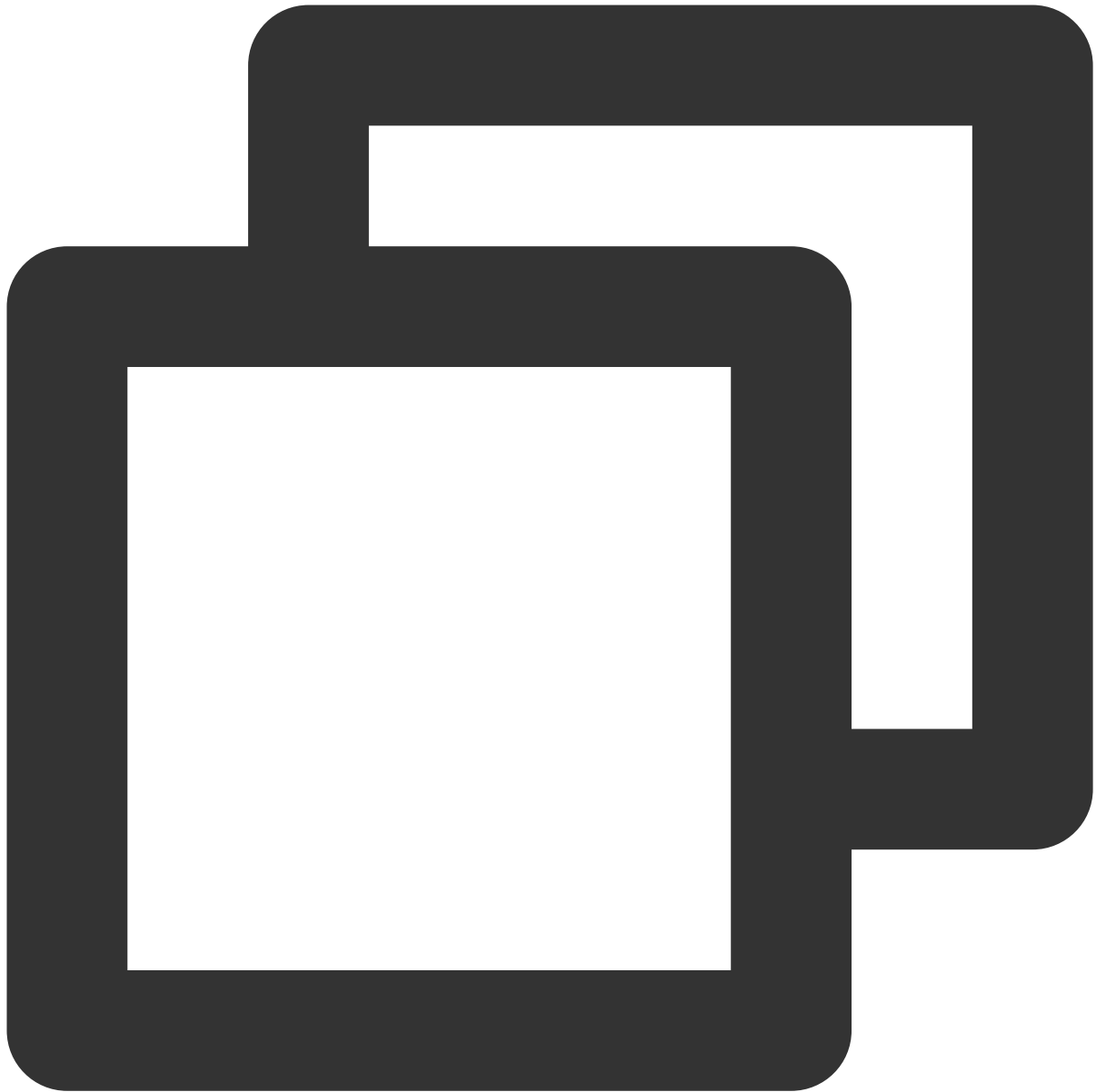
## Troubleshooting

1. Log in to the source database.
2. Change the value of `innodb_stats_on_metadata` to `OFF` .



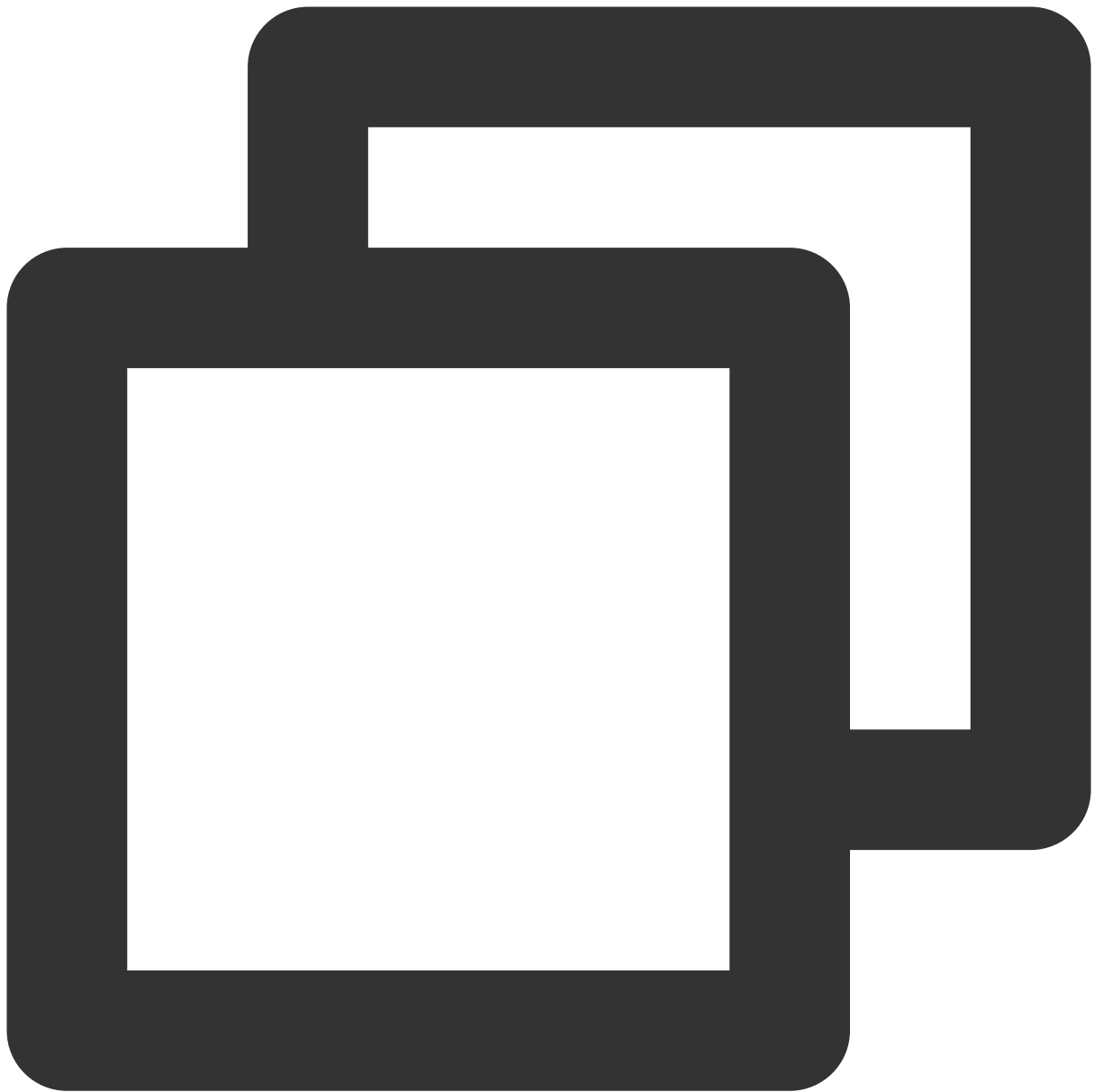
```
set global innodb_stats_on_metadata = OFF;
```

3. Check whether the configuration takes effect.



```
show global variables like '%innodb_stats_on_metadata%';
```

The system should display a result similar to the following:



```
mysql> show global variables like '%innodb_stats_on_metadata%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| innodb_stats_on_metadata | OFF   |
+-----+-----+
1 row in set (0.00 sec)
```

4. Run the verification task again.

# Version Check

Last updated : 2024-08-01 14:45:50

## Check Details

Log in to the database, run `select tbase_version();`, and `Tbase_V2.xx` must be returned.

## Troubleshooting

Select the ID of the TDSQL for PostgreSQL instance on the supported version.



# Source Instance Permission Check

Last updated : 2024-07-08 16:55:54

## Check Details

Check whether you have the operation permissions of the database by referring to the following:

[Permission requirements for data migration](#)

[Permission requirements for data sync](#)

[Permission requirements for data subscription](#)

## Troubleshooting

If you don't have the operation permissions, get authorized based on the permission requirements in the check details, and run the verification task again.

# Binlog Parameter Check

Last updated : 2024-07-08 16:56:38

## Check Details

You need to configure the source database's binlog parameters in compliance with the following requirements. If the verification fails, fix it as instructed in this document.

The `log_bin` variable must be set to `ON` .

The `binlog_format` variable must be set to `ROW` .

`binlog_row_image` must be set to `FULL` .

If the source database is MySQL 5.6 or later, `gtid_mode` can only be set to `ON` or `OFF` . We recommend that you set it to `ON` , because if it is set to `OFF` , an alarm will be triggered, and if it is set to `ON_PERMISSIVE` or `OFF_PERMISSIVE` , an error will be reported.

The `server_id` parameter must be set manually and cannot be `0` .

It is not allowed to set `do_db` and `ignore_db` .

If the source database is a replica database, the `log_slave_updates` variable must be set to `ON` .

We recommend that you retain the binlog of the source database for at least three days; otherwise, the task cannot be resumed from the checkpoint and will fail.

## Troubleshooting

### Enabling binlog

`log_bin` controls the binlog switch. You need to enable binlog to log all database table structure and data changes.

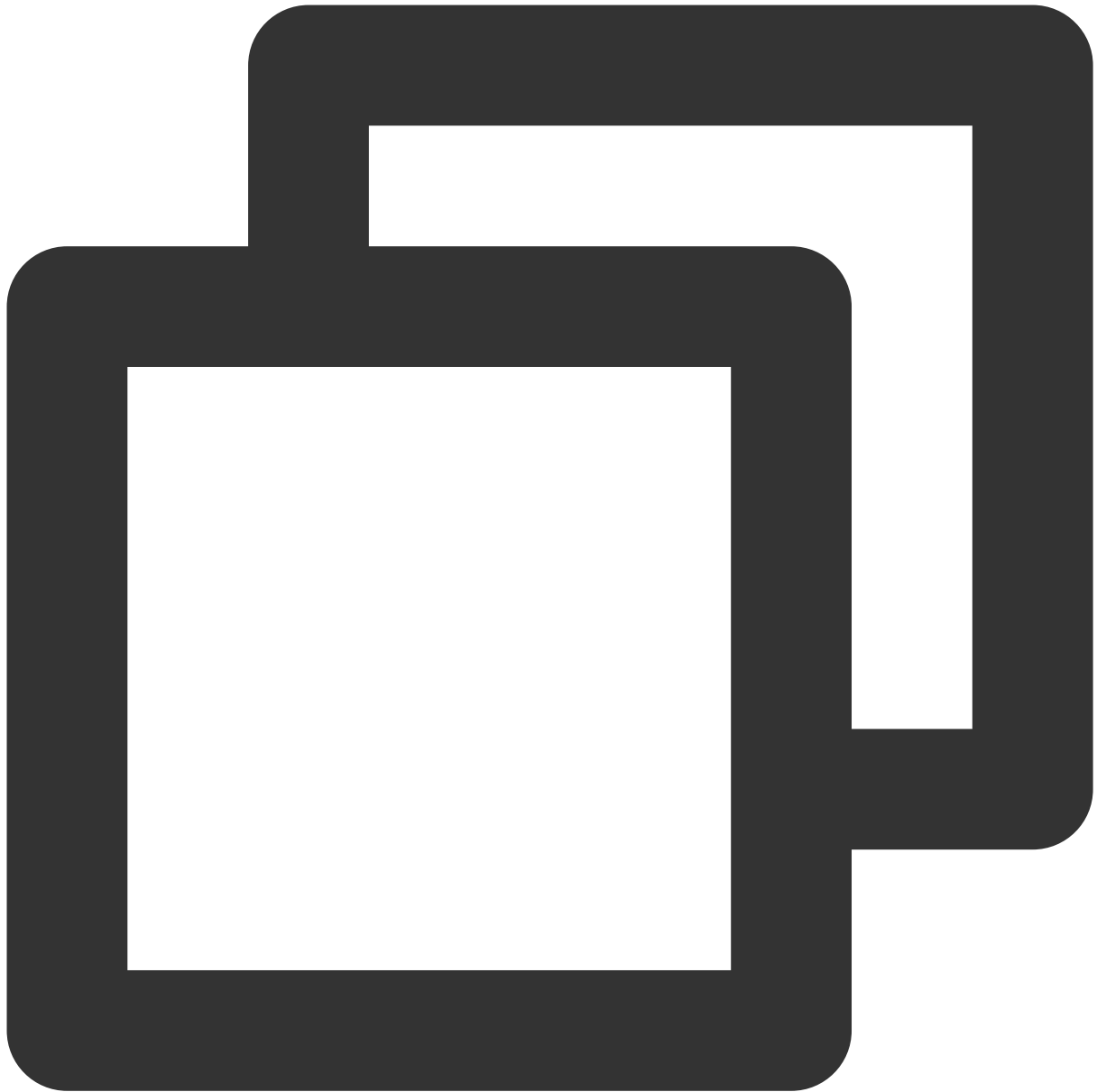
If a similar error occurs, fix it as follows:

1. Log in to the source database.
2. Modify the `my.cnf` configuration file of the source database as follows:

The modification of the `log_bin` parameter only takes effect after the database is restarted. Therefore, if errors are also reported for the `binlog_format` , `server_id` , `binlog_row_image` and `expire_logs_days` parameters in the verification stage, we recommend that you modify all these parameters before restarting the database so that all the modifications can take effect.

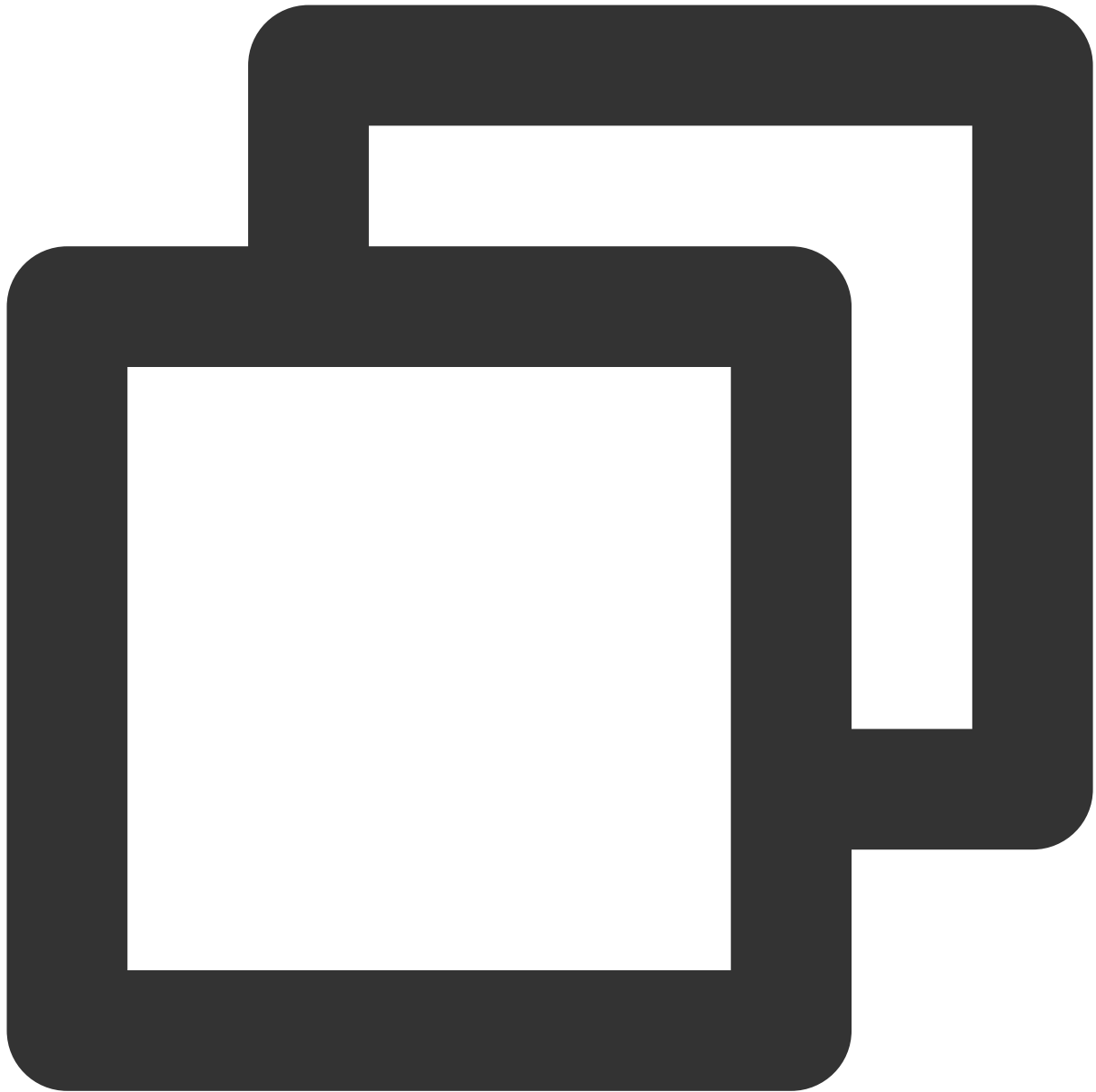
### Note

The default path of the `my.cnf` configuration file is `/etc/my.cnf` , subject to the actual conditions.



```
log_bin = MYSQL_BIN
binlog_format = ROW
server_id = 2      //We recommend that you set it to an integer above 1. The value h
binlog_row_image = FULL
expire_logs_days = 3      //Modify the binlog retention period (at least 3 days pref
```

3. Run the following command to restart the source database:

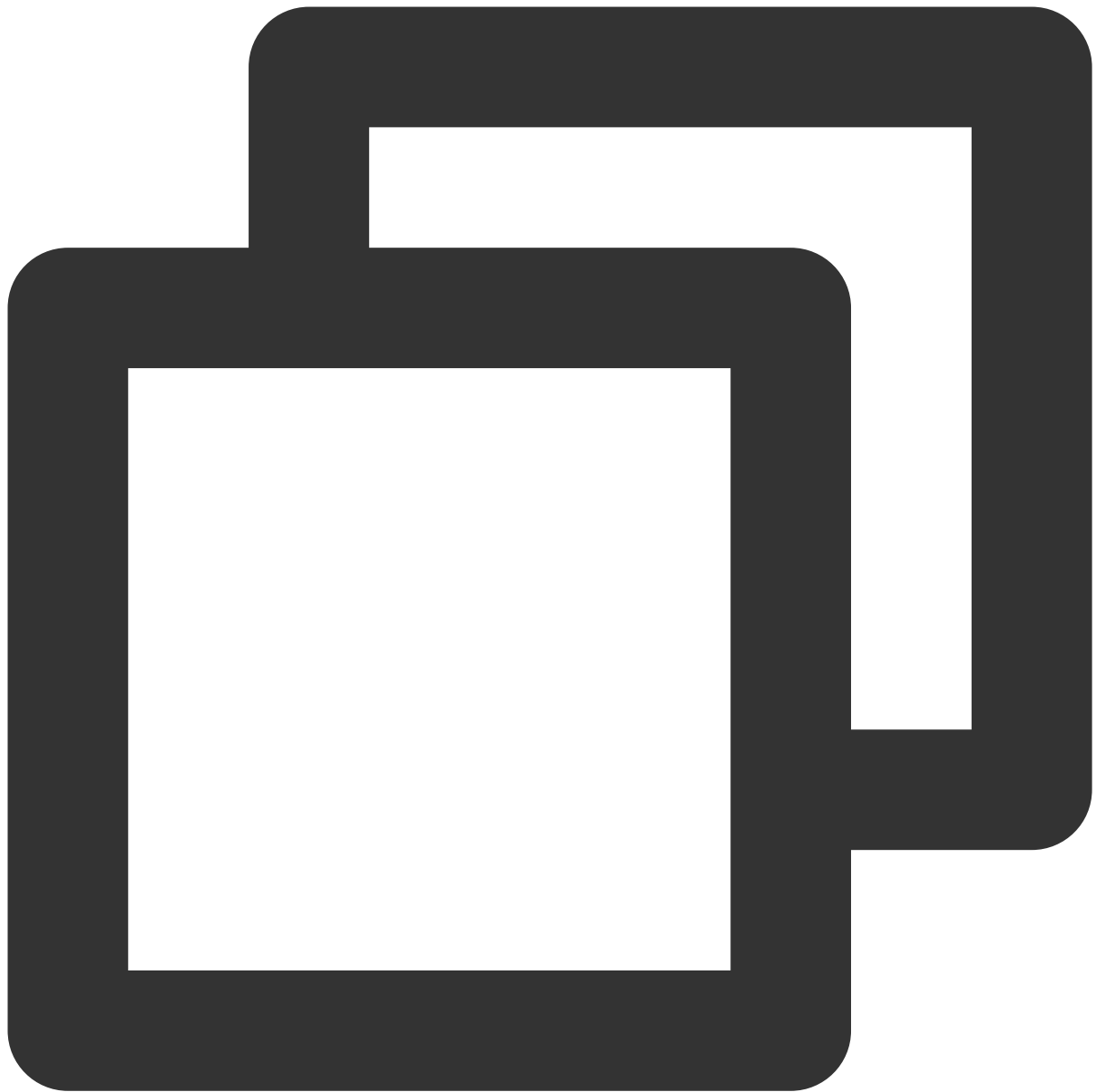


```
[$Mysql_Dir]/bin/mysqladmin -u root -p shutdown  
[$Mysql_Dir]/bin/safe_mysqld &
```

**Note**

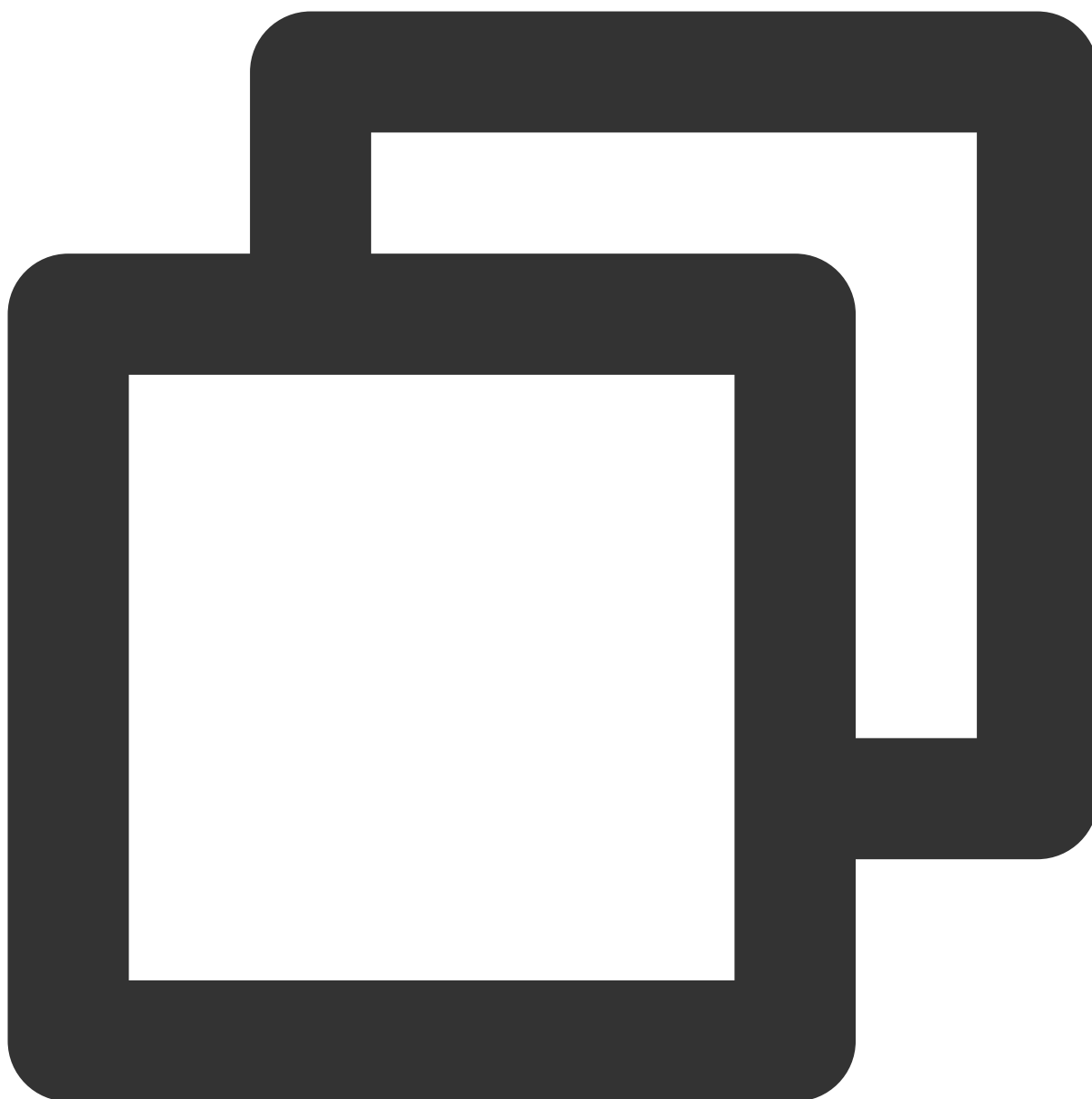
[ \$Mysql\_Dir] is the installation path of the source database. Replace it with the actual path.

4. Check whether the binlog feature has been enabled.



```
show variables like '%log_bin%';
```

The system will display a result similar to the following:



```
mysql> show variables like '%log_bin%';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| log_bin       | ON    |
+-----+-----+
| binlog_format | ROW   |
+-----+-----+
| binlog_row_image | FULL |
+-----+-----+
1 row in set (0.00 sec)
```

5. Run the verification task again.

## Modifying `binlog_format` parameter

`binlog_format` specifies one of the following three binlog formats:

**STATEMENT** : Each SQL statement that modifies the data will be logged into the binlog of the source/primary database. When replicating data, the replica/secondary database will run the same SQL statements as those in the source/primary database. This format can reduce the binlog size. However, the replica/secondary database may not be able to properly replicate certain functions.

**ROW** : The binlog will log the modification of each data row, and the replica/secondary database will modify the same data. This format guarantees the correct source-replica or primary-secondary replication, but the binlog size will increase.

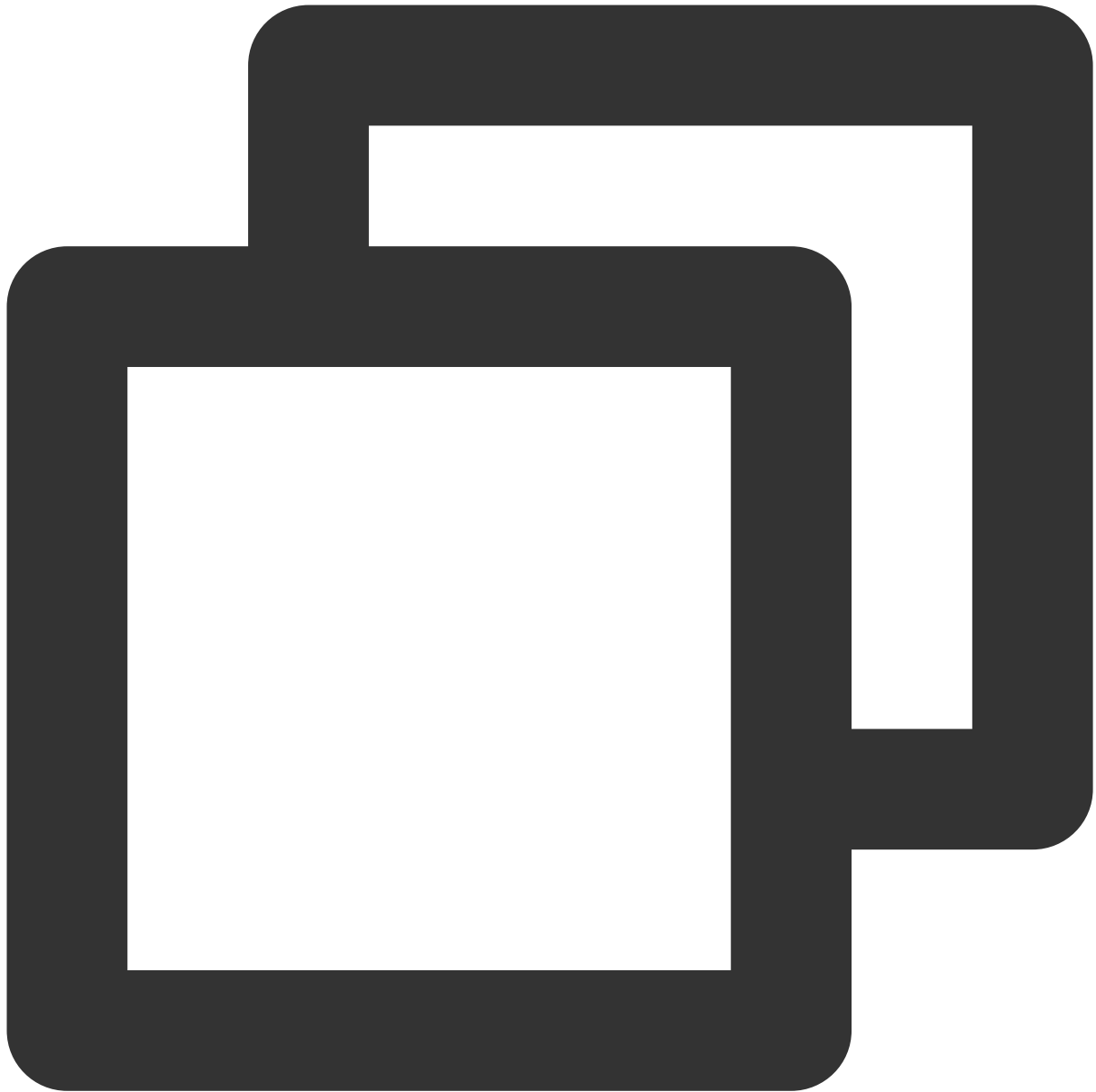
**MIXED** : It is a combination of the above two formats. MySQL will automatically select **STATEMENT** or **ROW** format to log each executed SQL statement.

Therefore, to ensure the correct source-replica or primary-secondary replication, the `binlog_format` parameter should be set to **ROW** . If a similar error occurs, fix it as follows:

### Note

Changes to this parameter can only take effect after all connections to the database are reset. If the source database is a replica/secondary database, you also need to restart the source-replica or primary-replica sync SQL thread to prevent current business connections from continuing writing data in the mode before modification.

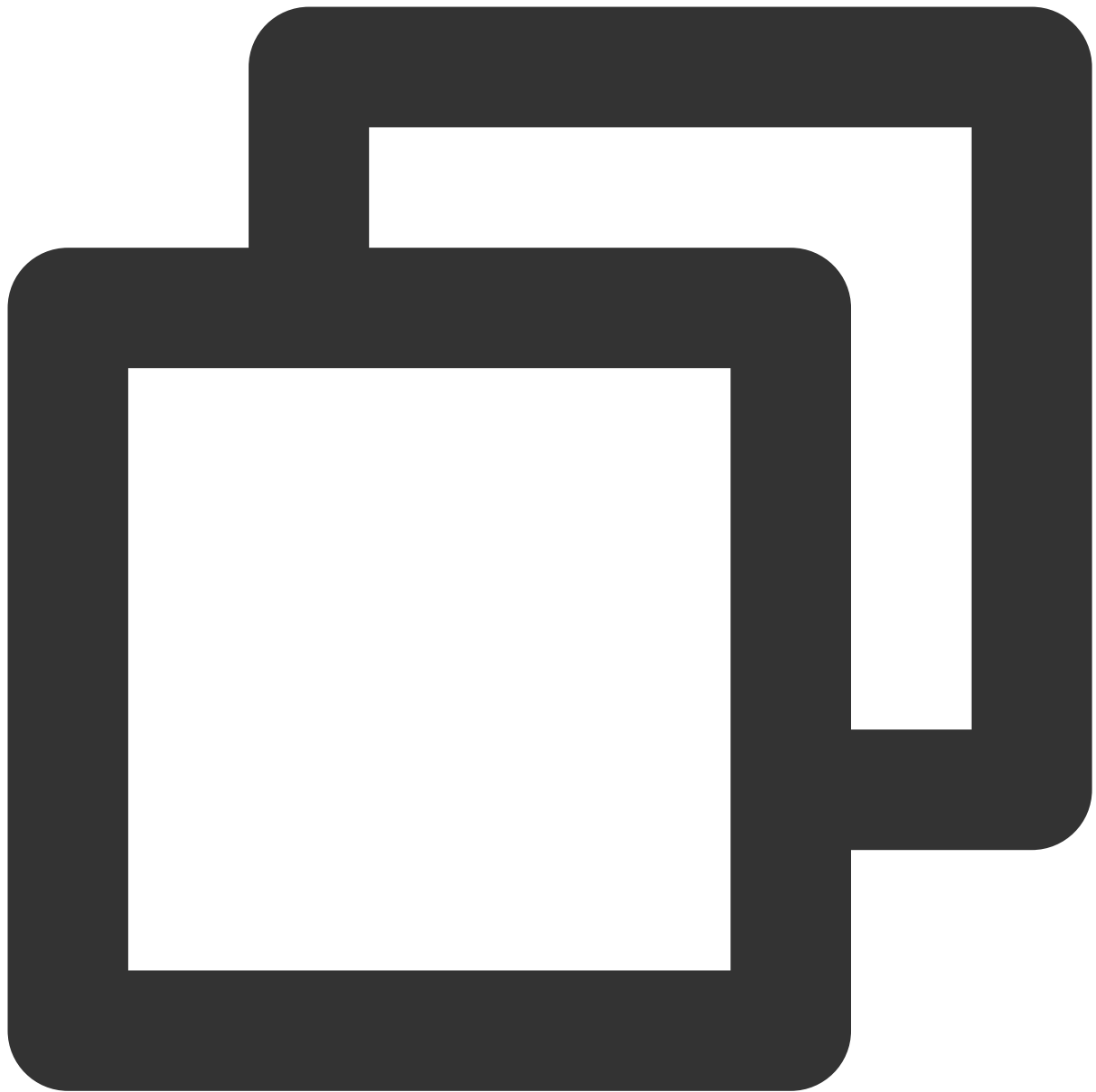
1. Log in to the source database.
2. Run the following command to modify `binlog_format` .



```
set global binlog_format = ROW;
```

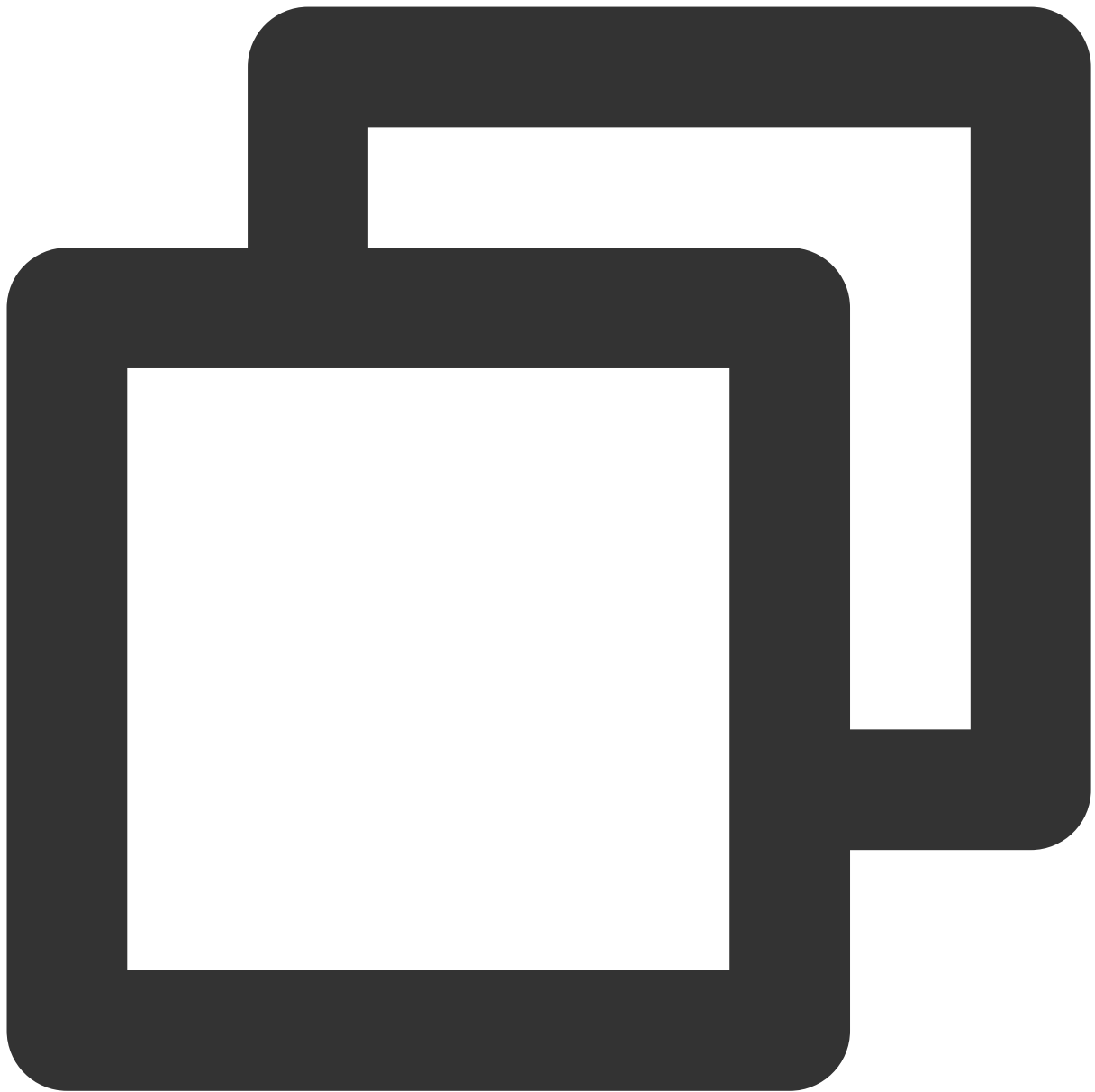
3. Restart the thread for the configuration to take effect. Then, run the following command to check whether the parameter modification takes effect:





```
show variables like '%binlog_format%';
```

The system will display a result similar to the following:



```
mysql> show variables like '%binlog_format%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| binlog_format | ROW   |
+-----+-----+
1 row in set (0.00 sec)
```

4. Run the verification task again.

**Modifying `binlog_row_image` parameter**

The `binlog_row_image` parameter determines how the binlog logs the pre-image (content before modification) and post-image (content after modification), which directly affects features such as data flashback and source-replica or primary-replica replication.

The `binlog_row_image` parameter takes effect only if `binlog_format` is set to `ROW`. The following describes the effects of specific values:

`FULL` : In `ROW` format, binlog will log all the pre-image and post-image column data information.

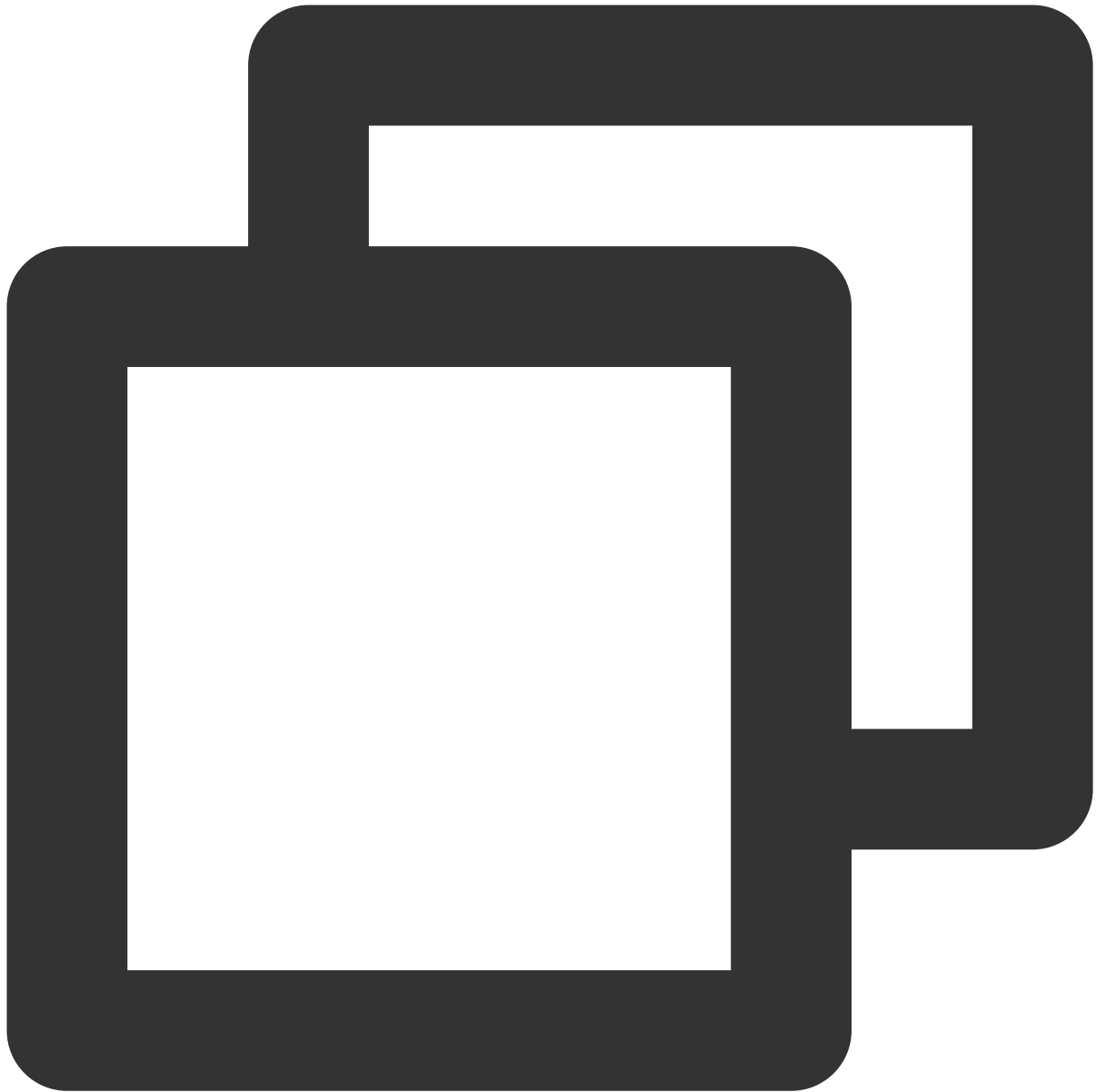
`MINIMAL` : In `ROW` format, if a table has no primary key or unique key, the pre-image will log all columns, and the post-image will log the modified columns. If it has a primary key or unique key, both the pre-image and post-image will only log the affected columns.

Therefore, you need to set `binlog_row_image` to `FULL` to make the source database binlog log the full image. If an error occurs, troubleshoot as follows:

### Note

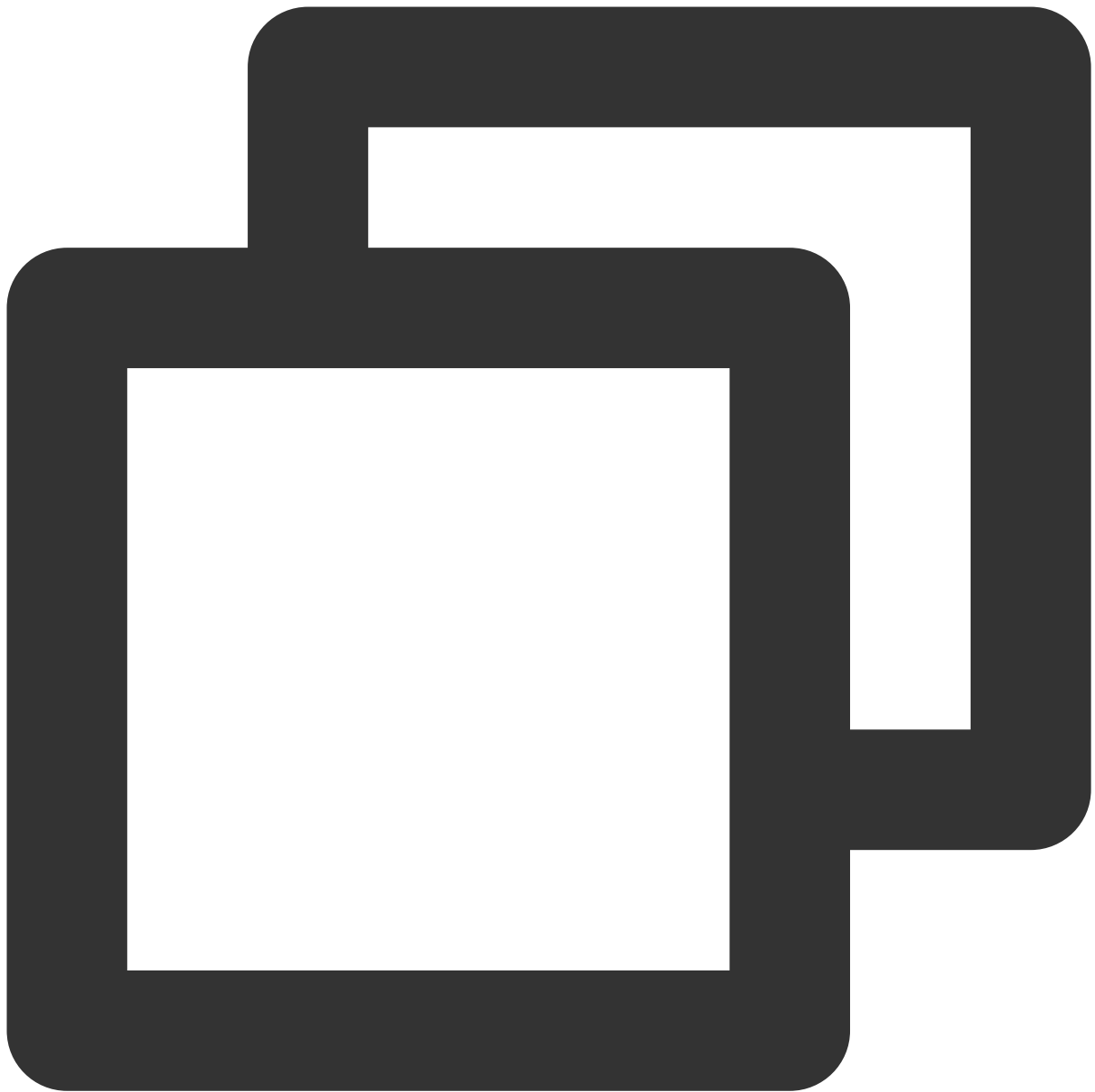
Changes to this parameter can only take effect after all connections to the database are reset. If the source database is a replica/secondary database, you also need to restart the source-replica or primary-replica sync SQL thread to prevent current business connections from continuing writing data in the mode before modification.

1. Log in to the source database.
2. Run the following command to modify `binlog_row_image` :



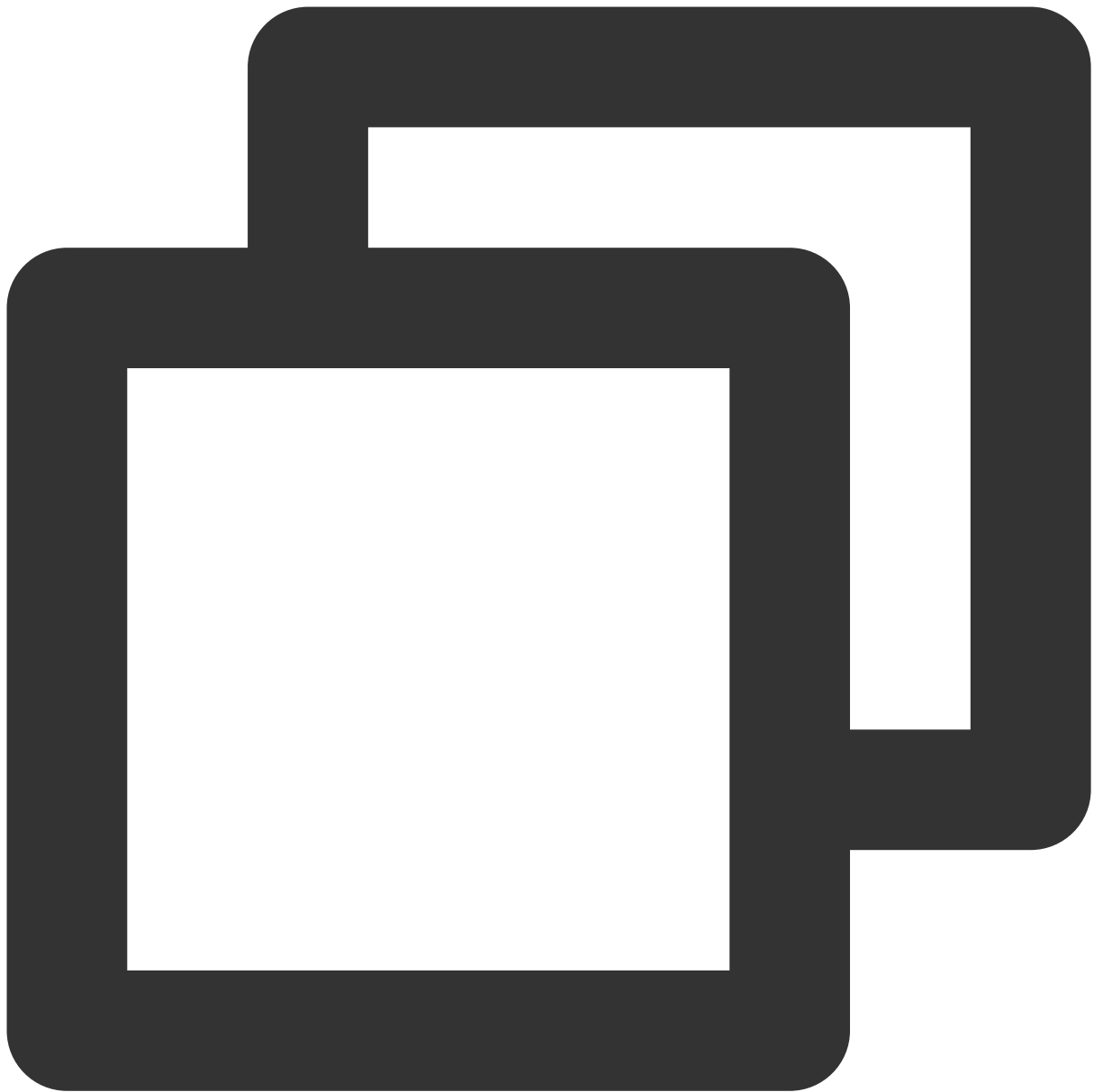
```
set global binlog_row_image = FULL;
```

3. Restart the thread for the configuration to take effect. Then, run the following command to check whether the parameter modification takes effect:



```
show variables like '%binlog_row_image%';
```

The system will display a result similar to the following:



```
mysql> show variables like '%binlog_row_image%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| binlog_row_image | FULL |
+-----+-----+
1 row in set (0.00 sec)
```

4. Run the verification task again.

### Modifying `gtid_mode` parameter

A global transaction identifier (GTID) uniquely identifies a transaction in the binlog. Using GTIDs can prevent disordered data or source-replica or primary-replica inconsistency due to repeated transaction executions. GTID is a new feature on MySQL 5.6. Therefore, this problem may only occur on MySQL 5.6 or later versions. DTS only allows you to set `gtid_mode` to `ON` or `OFF`. We recommend that you set it to `ON`; otherwise, an alarm will be triggered during verification.

The alarm does not affect the migration or sync task but affects the business. After GTID is set, if HA switch occurs in the source database during incremental data sync, DTS will be switched and restarted, which is almost imperceptible to the task; if GTID is not set, the task will fail after disconnection and cannot be resumed.

Below are the valid values of `gtid_mode`. When modifying the value, you can only do so in the specified sequence step by step; for example, if you want to change `OFF` to `ON`, you should modify the `gtid_mode` value in the following sequence: `OFF` <-> `OFF_PERMISSIVE` <-> `ON_PERMISSIVE` <-> `ON`.

`OFF`: All new transactions in the source/primary database and all transactions in the replica/secondary database must be anonymous.

`OFF_PERMISSIVE`: All new transactions in the source/primary database must be anonymous. Transactions in the replica/secondary database can be anonymous or GTID transactions but cannot only be GTID transactions.

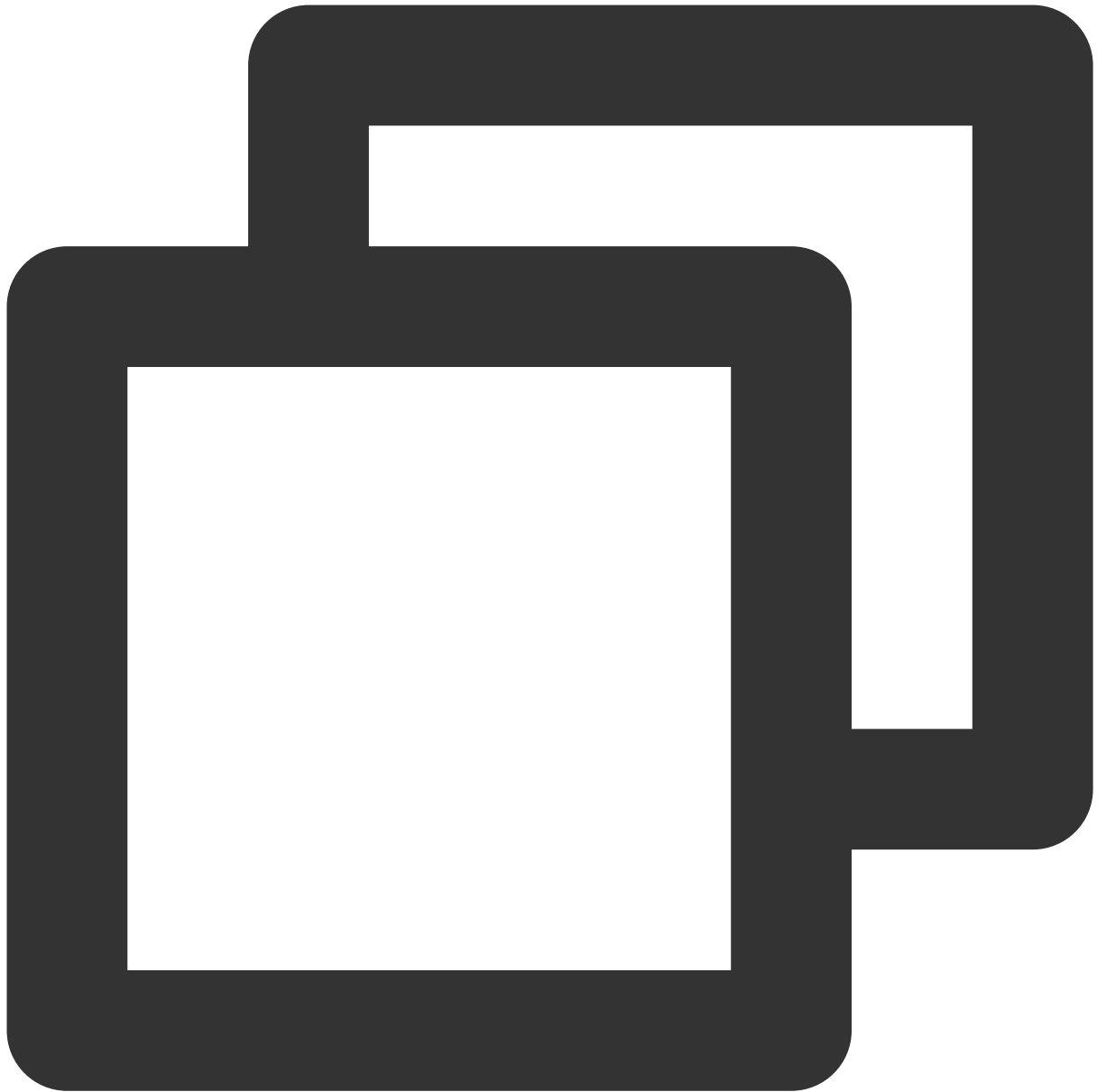
`ON_PERMISSIVE`: All new transactions in the source/primary database must be GTID transactions, and transactions in the replica/secondary database can be anonymous or GTID transactions.

`ON`: All new transactions in the source/primary database and all transactions in the replica/secondary database must be GTID transactions.

If a similar alarm is triggered, fix it as follows:

1. Log in to the source database.
2. Set `gtid_mode = OFF_PERMISSIVE` on the source/primary and replica databases.

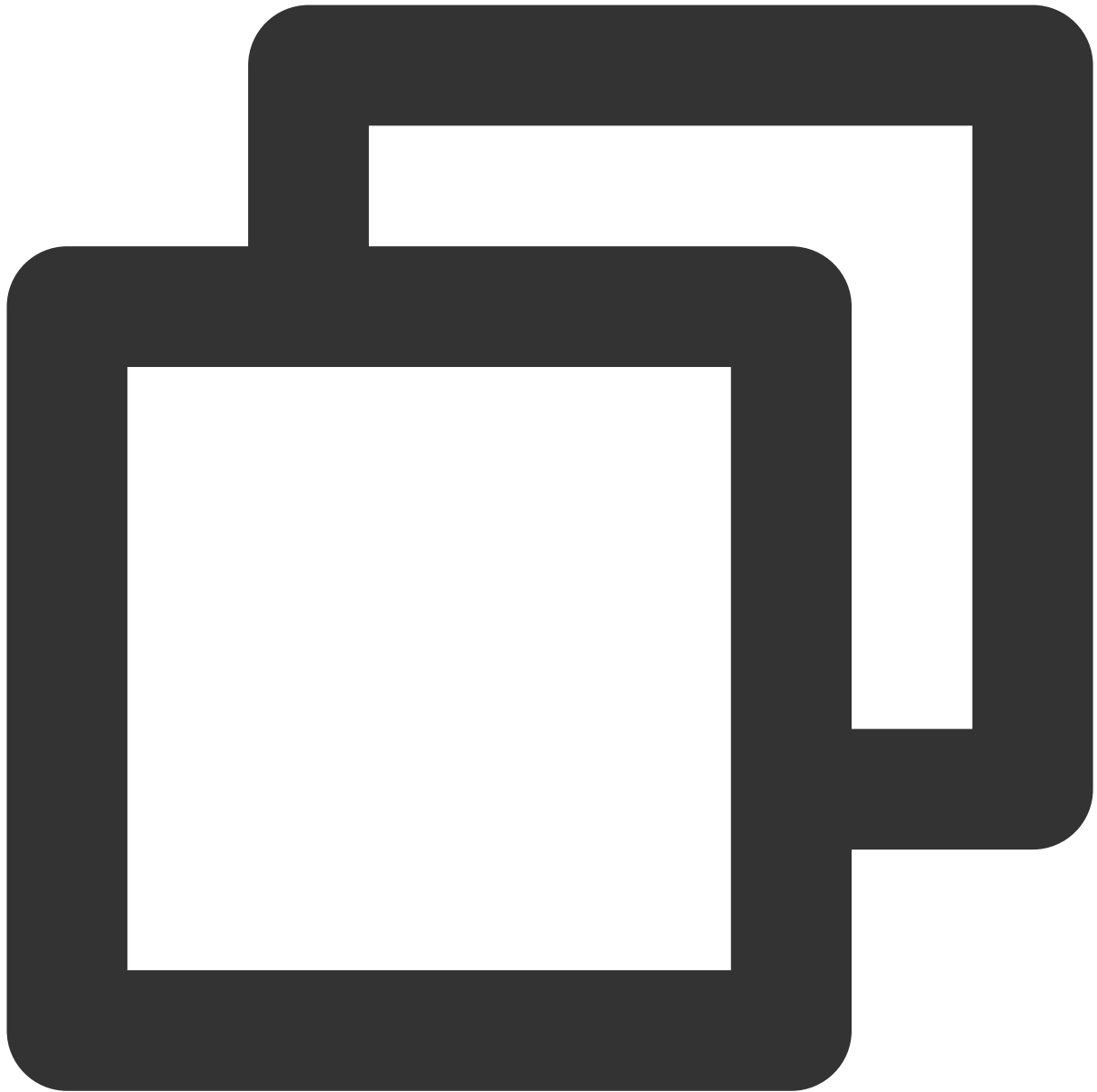
On MySQL versions earlier than v5.7.6, you need to modify the parameter in the `my.cnf` configuration file and restart the database to make the change take effect. On v5.7.6 and later, you can modify the parameter through global naming without restarting the database, but you must reset all business connections.



```
set global gtid_mode = OFF_PERMISSIVE;
```

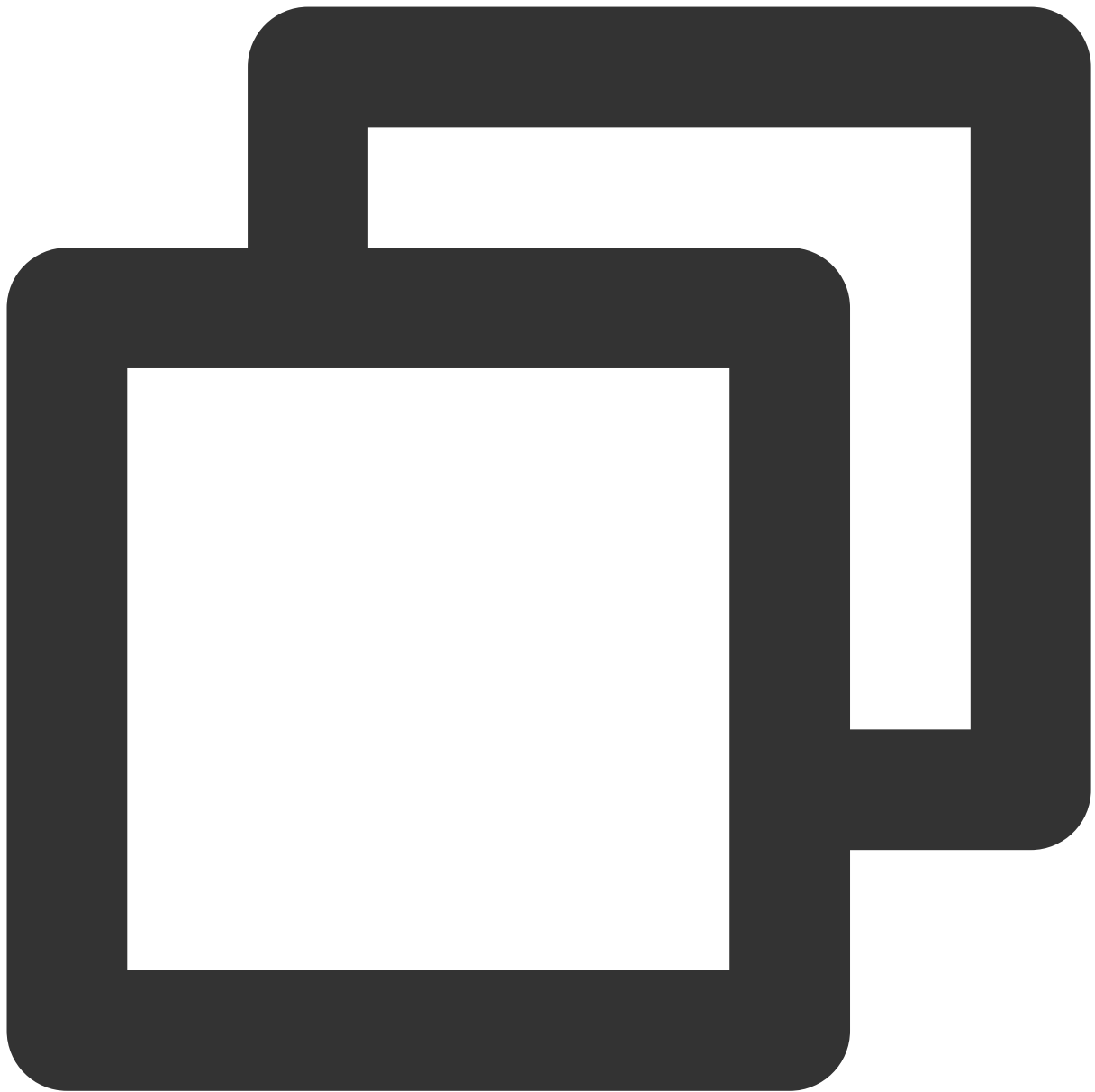
3. Set `gtid_mode = ON_PERMISSIVE` on the source/primary and replica databases.





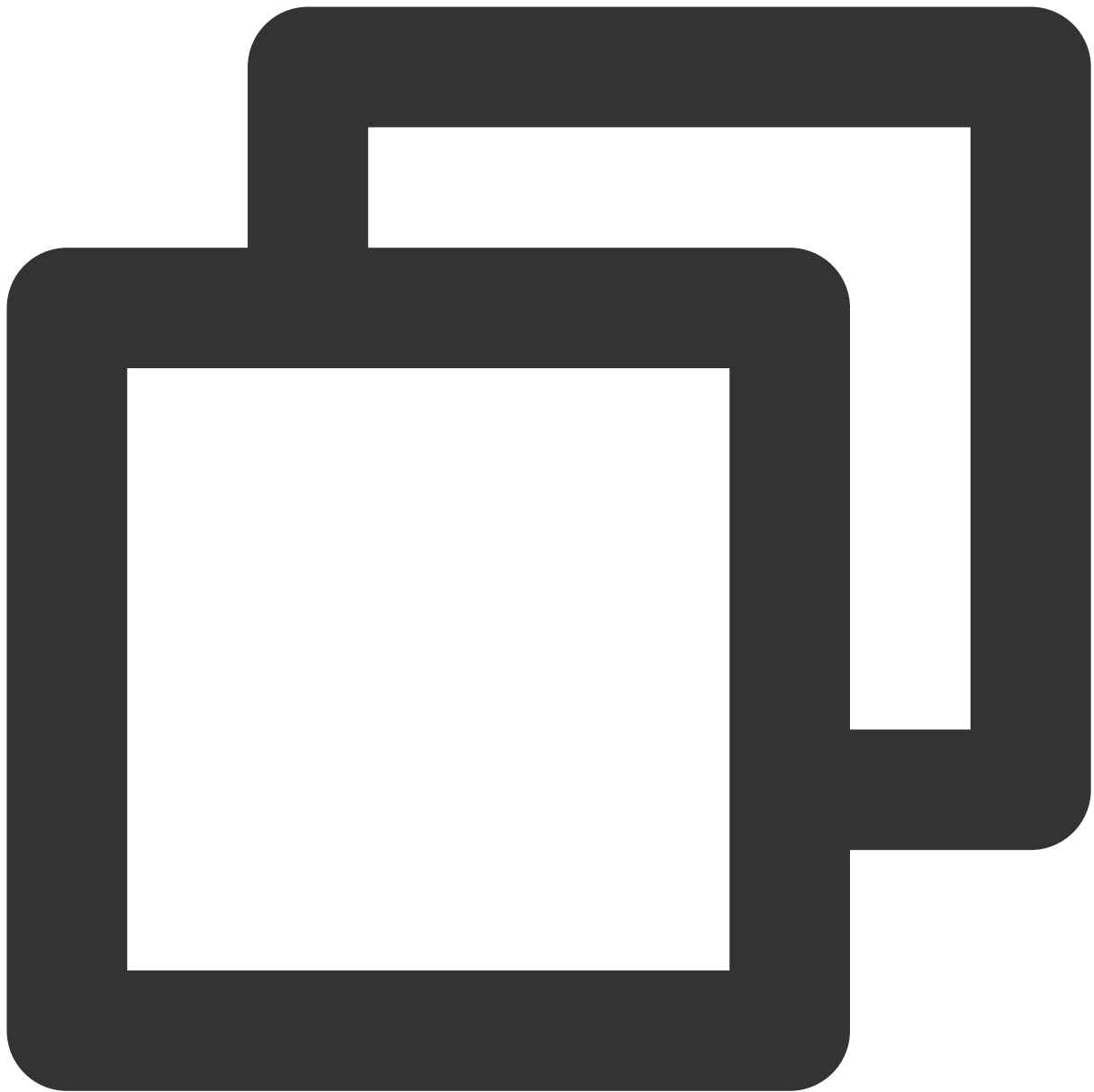
```
set global gtid_mode = ON_PERMISSIVE;
```

4. Run the following command on each instance node to check whether consumption of anonymous transactions is completed. If the parameter value is `0`, the consumption is completed.



```
show variables like '%ONGOING_ANONYMOUS_TRANSACTION_COUNT%';
```

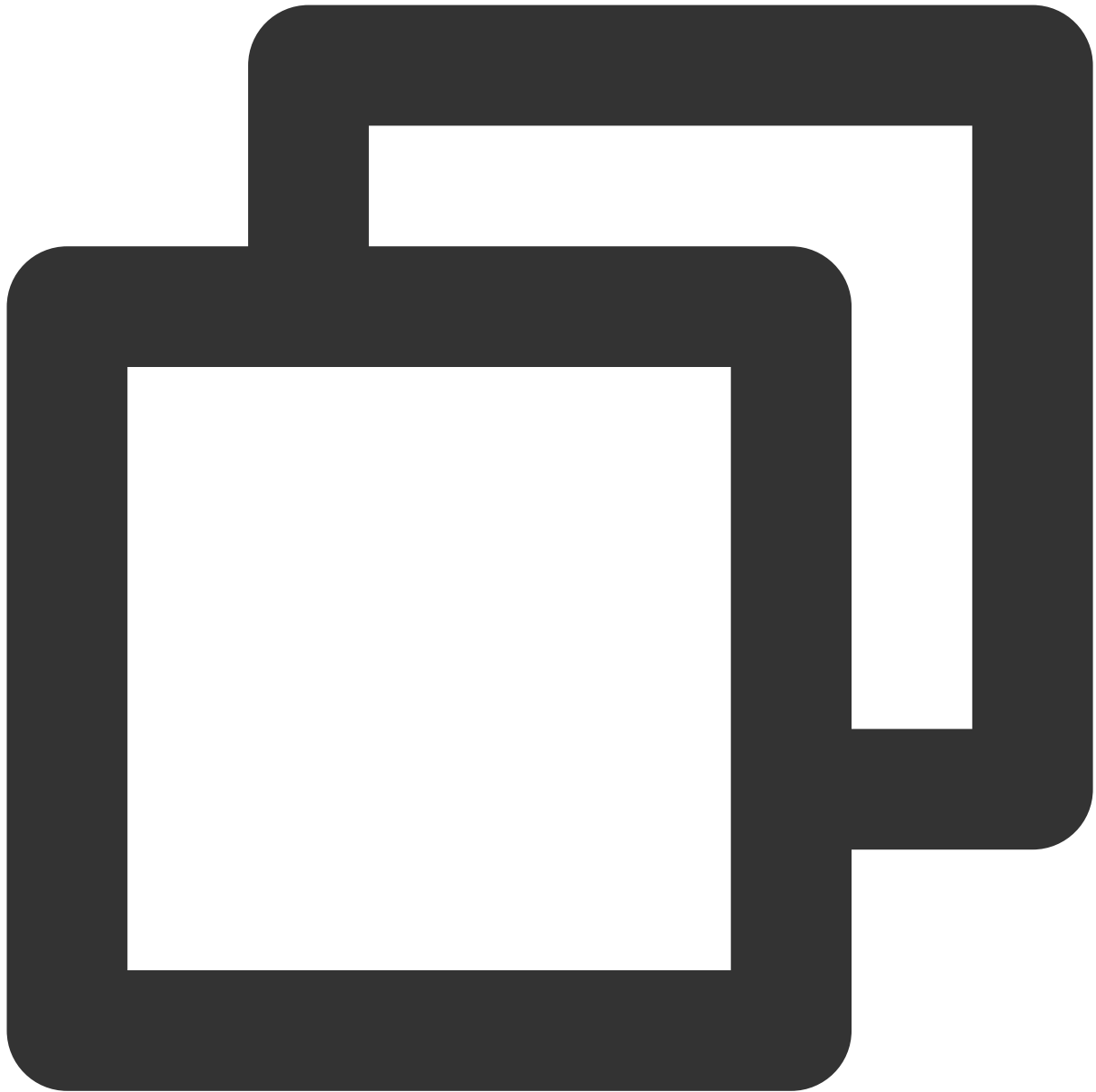
The system will display a result similar to the following:



```
mysql> show variables like '%ONGOING_ANONYMOUS_TRANSACTION_COUNT%';
```

```
+-----+-----+
| Variable_name                | Value |
+-----+-----+
| Ongoing_anonymous_transaction_count | 0      |
+-----+-----+
1 row in set (0.00 sec)
```

5. Set `gtid_mode = ON` on the source/primary and replica databases.

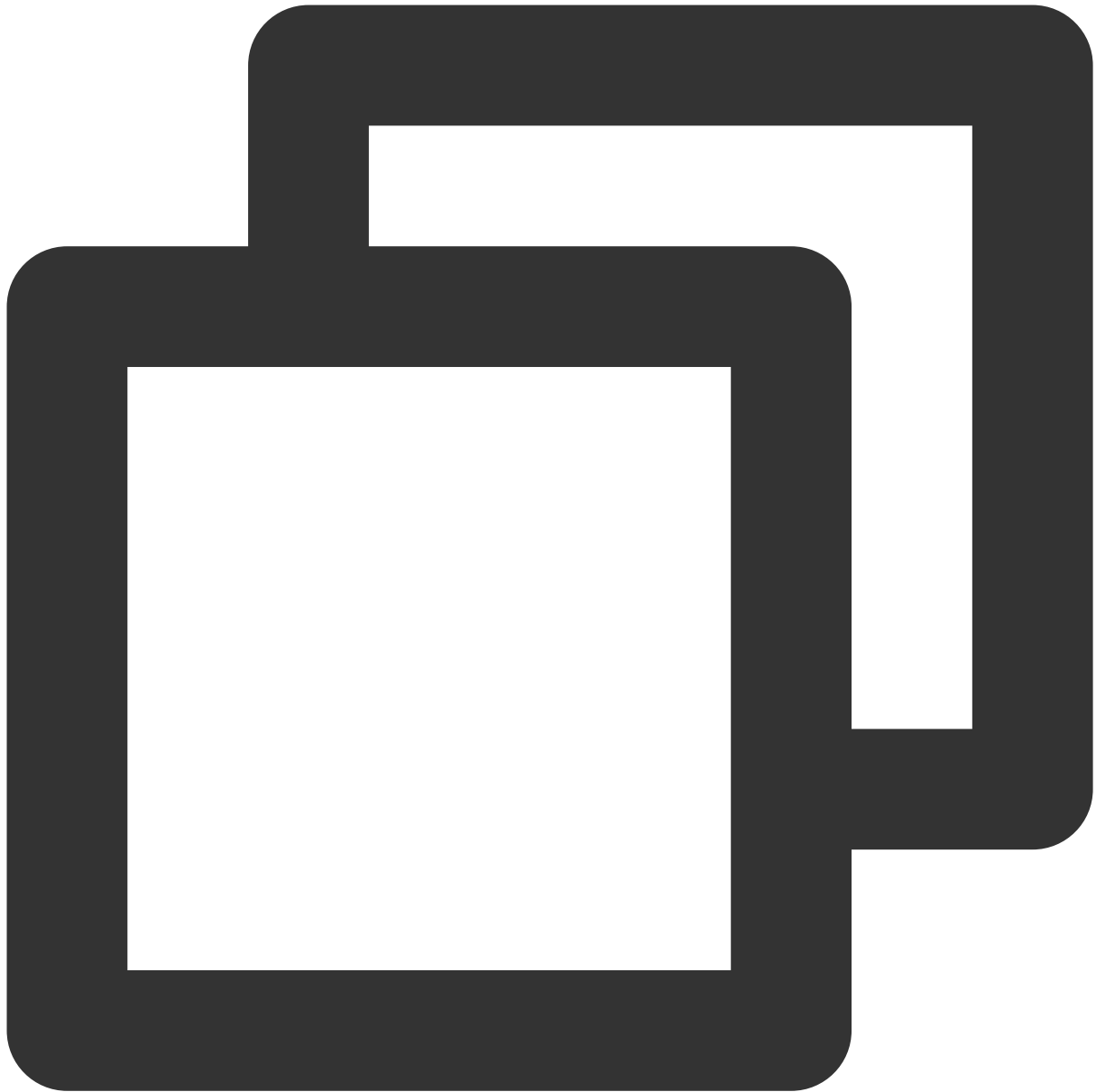


```
set global gtid_mode = ON;
```

6. Add the following content to the `my.cnf` file and restart the database to make the initial values take effect.

**Note**

The default path of the `my.cnf` configuration file is `/etc/my.cnf`, subject to the actual conditions.



```
gtid_mode = on
enforce_gtid_consistency = on
```

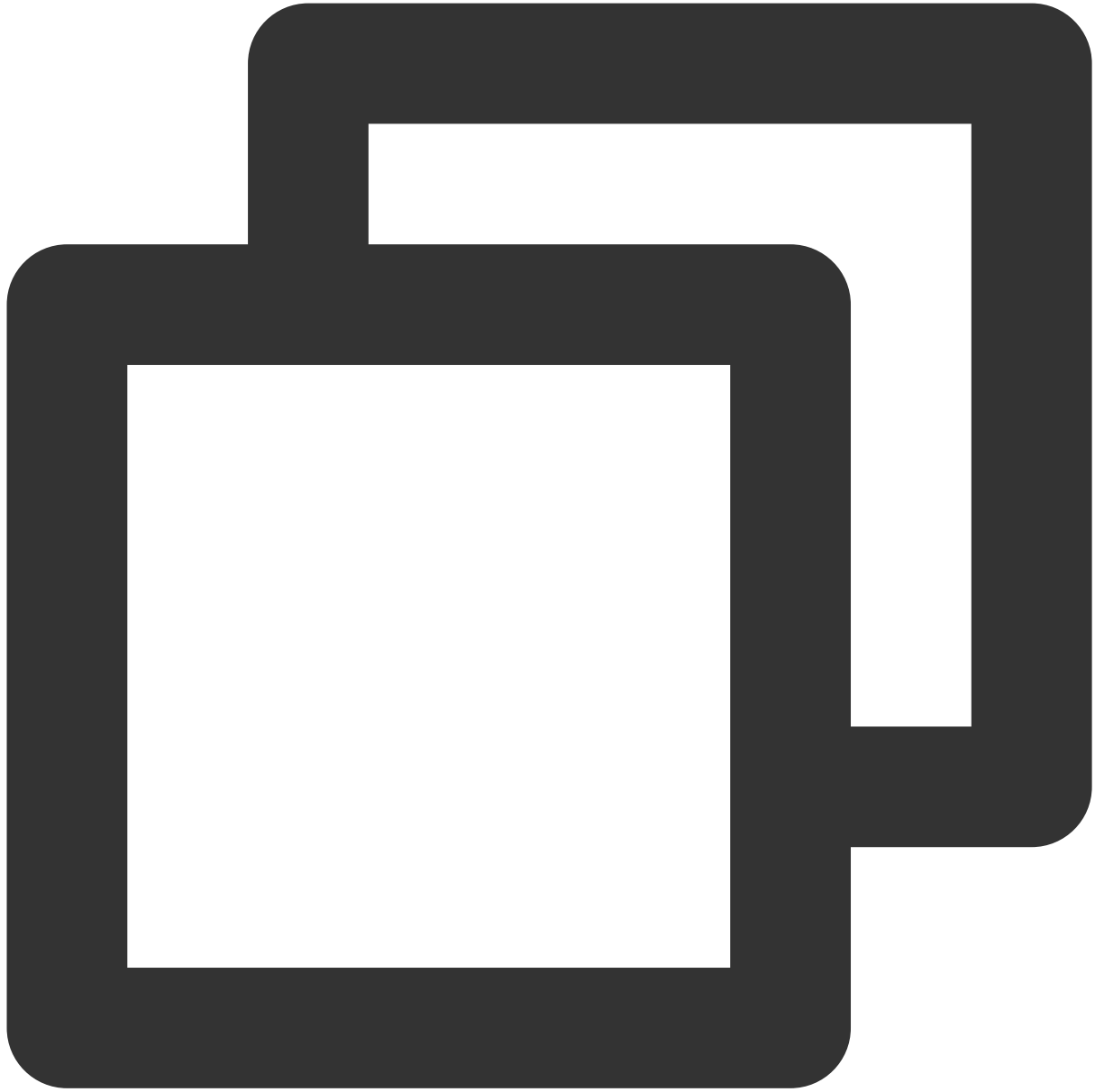
7. Run the verification task again.

### Modifying `server_id` parameter

The `server_id` parameter must be set manually and cannot be `0`. The default value of this parameter is `1`, but the configuration may not be correct even if the queried parameter value is `1`. You still need to set it manually.

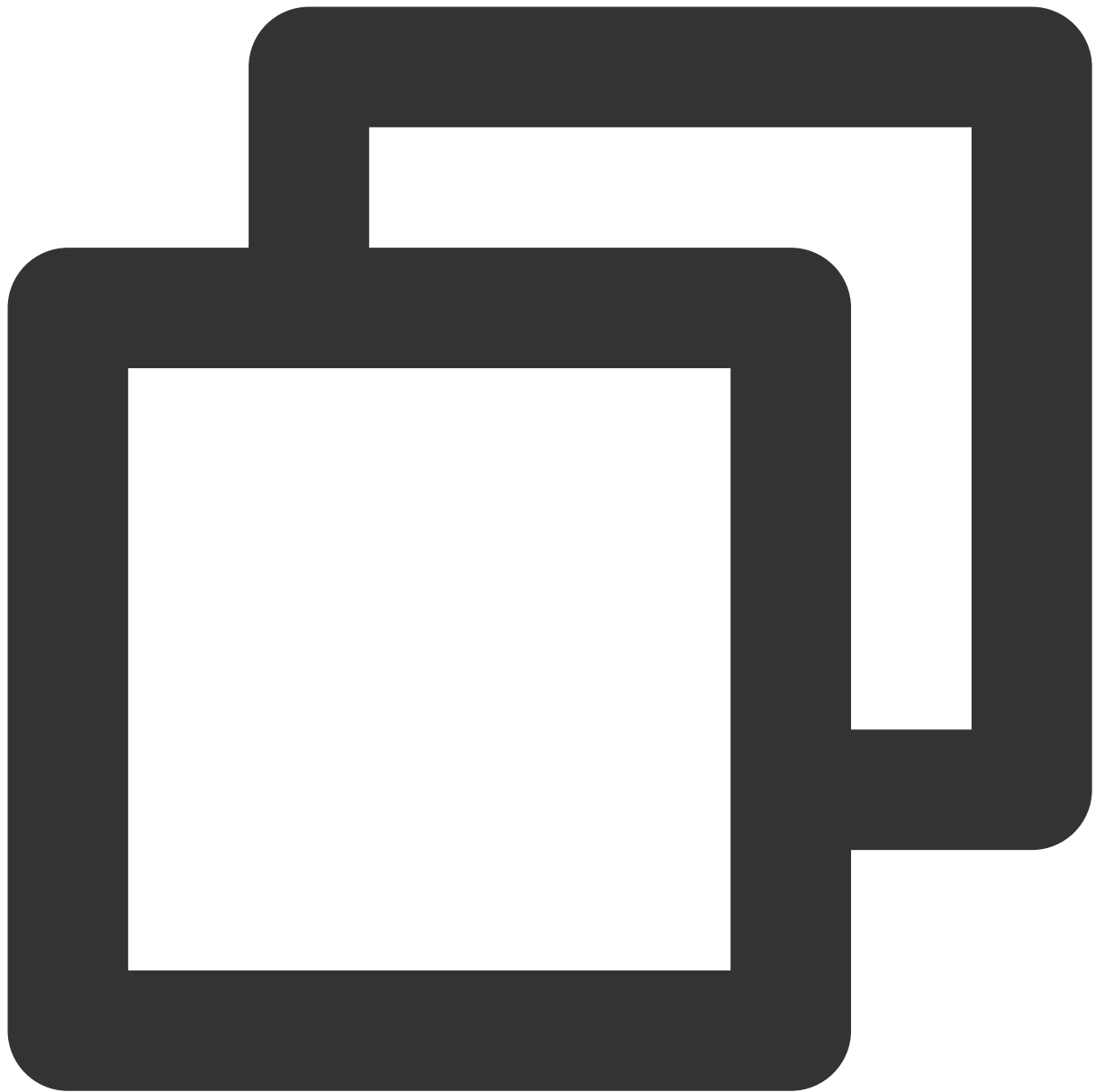
1. Log in to the source database.

2. Run the following command to modify `server_id` :



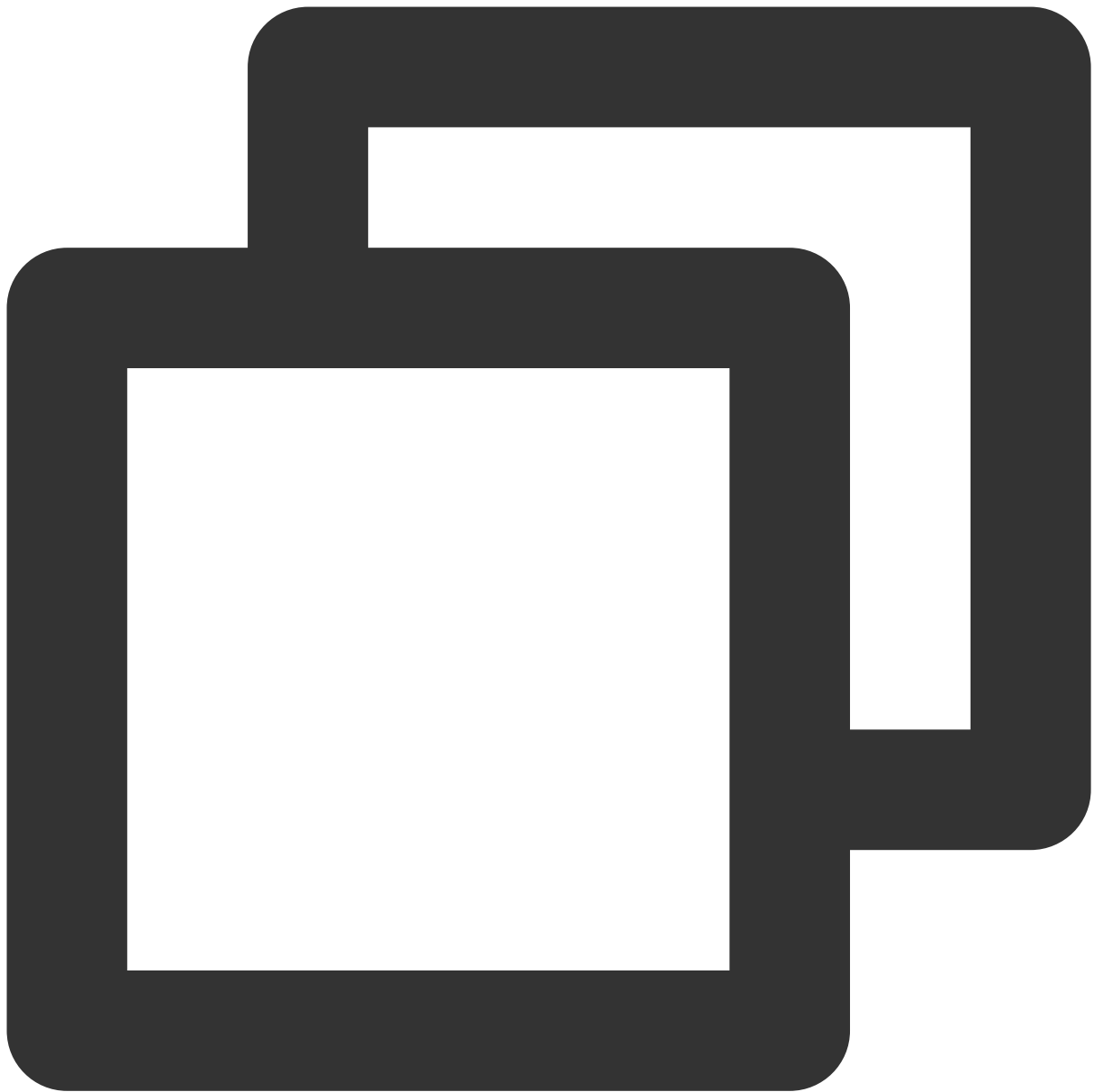
```
set global server_id = 2; // We recommend that you set it to an integer above 1. T
```

3. Run the following command to check whether the parameter modification takes effect:



```
show global variables like '%server_id%';
```

The system will display a result similar to the following:



```
mysql> show global variables like '%server_id%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| server_id     | 2     |
+-----+-----+
1 row in set (0.00 sec)
```

4. Run the verification task again.

**Deleting `do_db` and `ignore_db` settings**



The binlog logs all executed DDL and DML statements in the database, while `do_db` and `ignore_db` are used to set the filter conditions for binlog.

`binlog_do_db` : Only the specified databases will be logged in the binlog (all databases will be logged by default).

`binlog_ignore_db` : The specified databases will not be logged in the binlog.

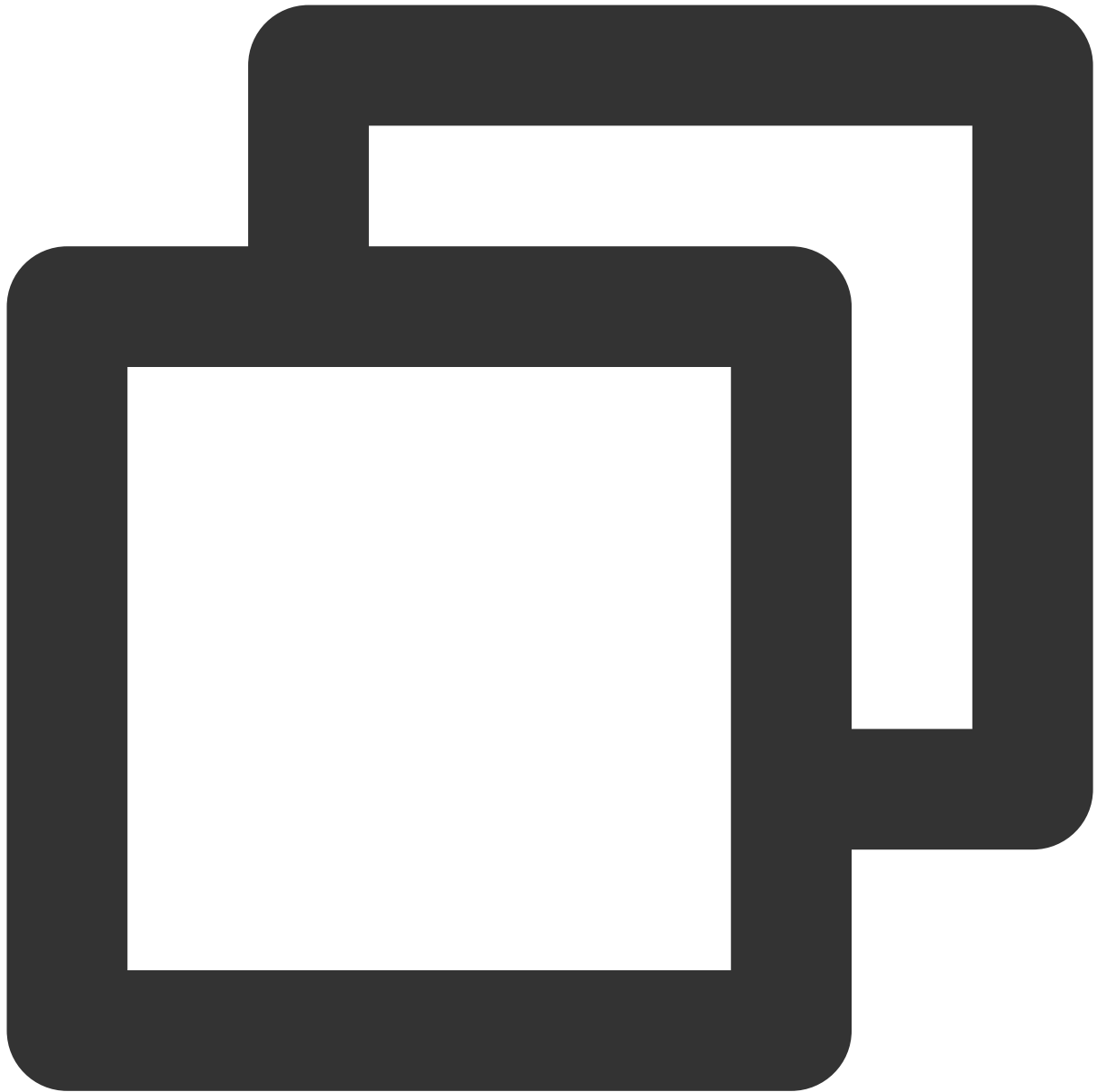
After `do_db` and `ignore_db` are set, some cross-database operations will not be logged in the binlog, and source-replica or primary-replica replication will be abnormal; therefore, this setting is not recommended. If a similar error occurs, fix it as follows:

1. Log in to the source database.
2. Modify the `my.cnf` configuration file in the source database to delete `do_db` and `ignore_db` settings.

#### Note

The default path of the `my.cnf` configuration file is `/etc/my.cnf`, subject to the actual conditions.

3. Run the following command to restart the source database:

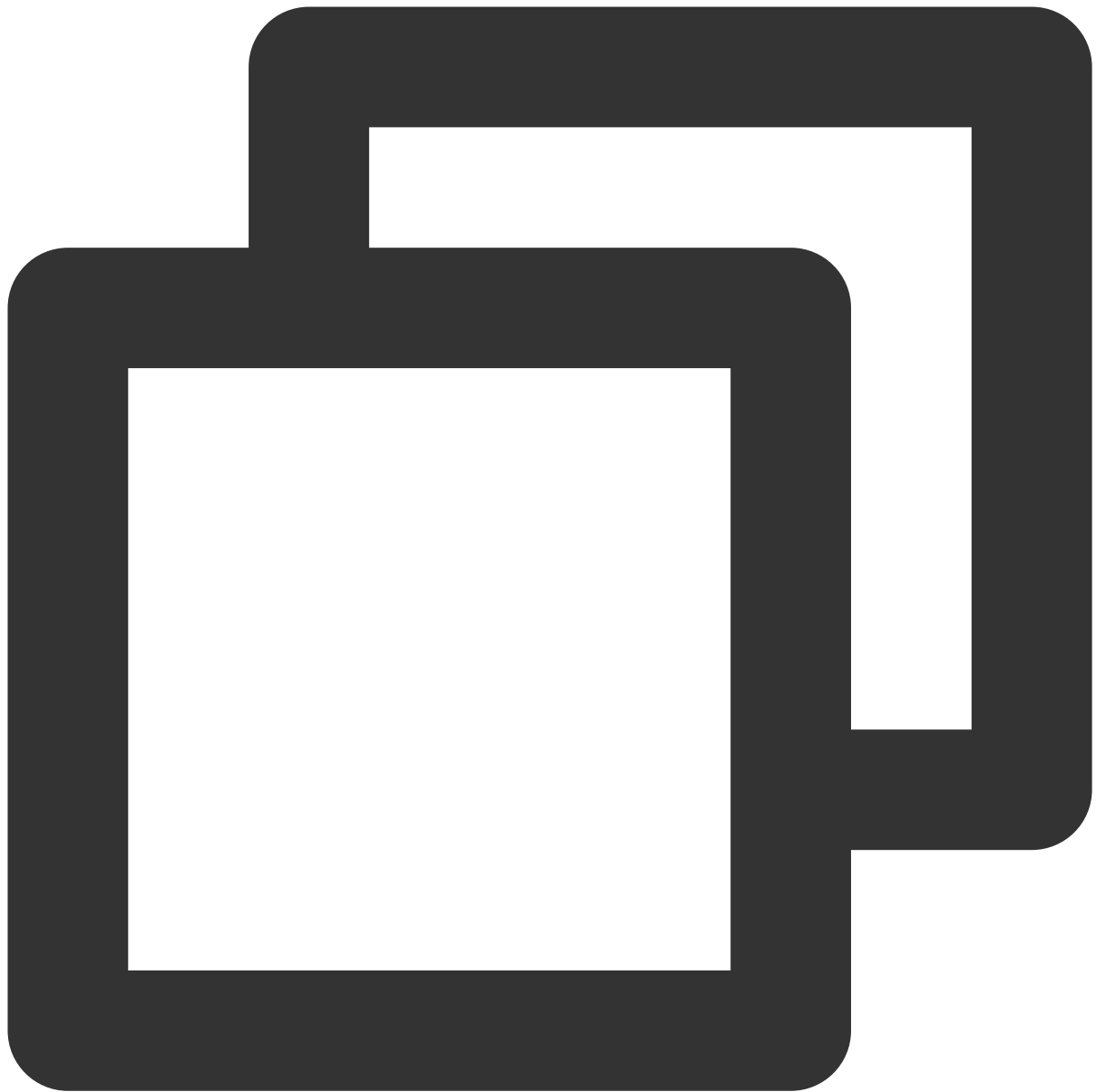


```
[$Mysql_Dir]/bin/mysqladmin -u root -p shutdown  
[$Mysql_Dir]/bin/safe_mysqld &
```

**Note**

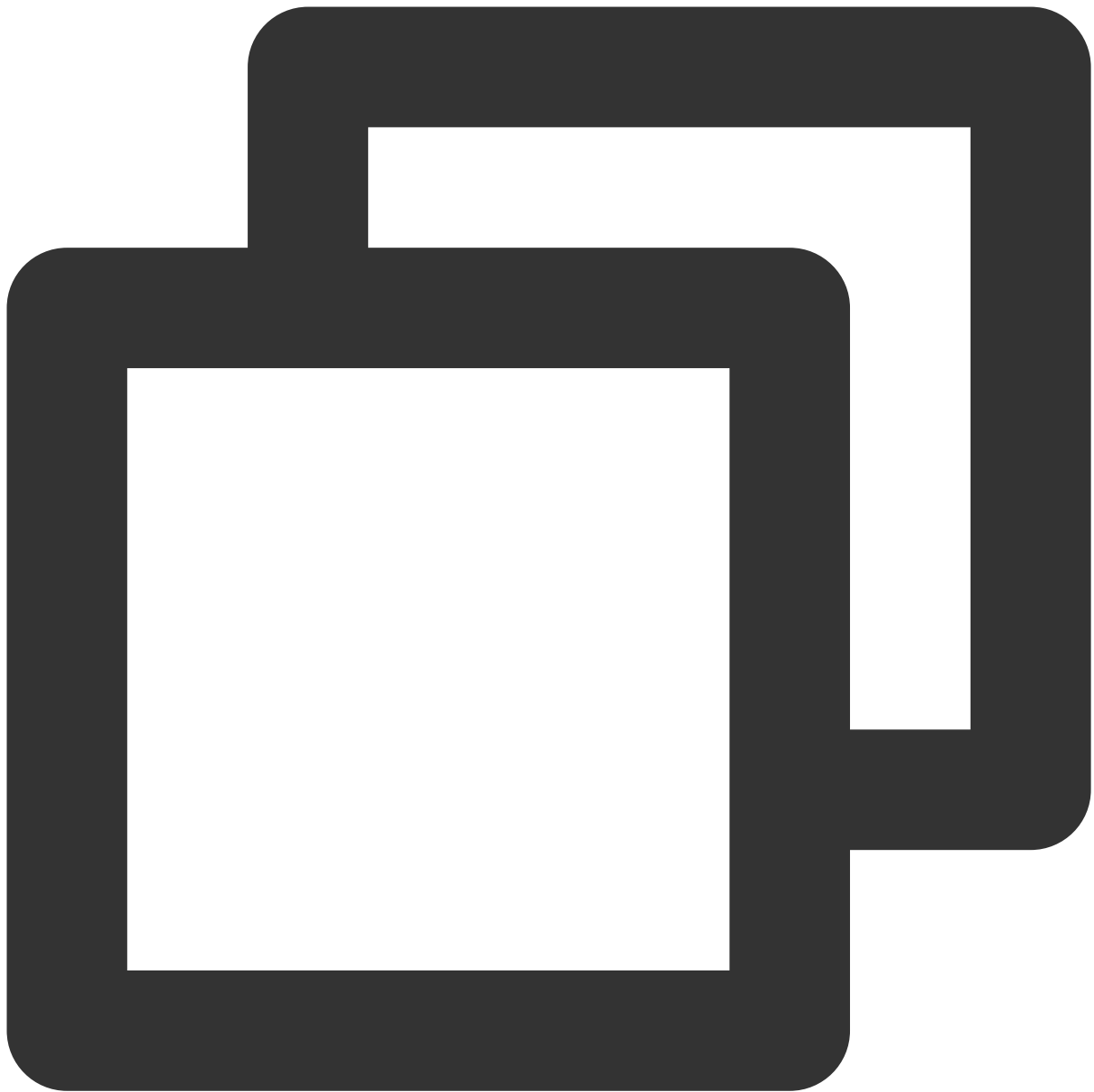
[ \$Mysql\_Dir ] is the installation path of the source database. Replace it with the actual path.

4. Check whether the parameter modification takes effect.



```
show master status;
```

The system will display a result similar to the following:



```
mysql> show master status;
```

File	Position	Binlog_Do_DB	Binlog_Ignore_DB	Executed_Gtid_Set
binlog.000011	154			

5. Run the verification task again.

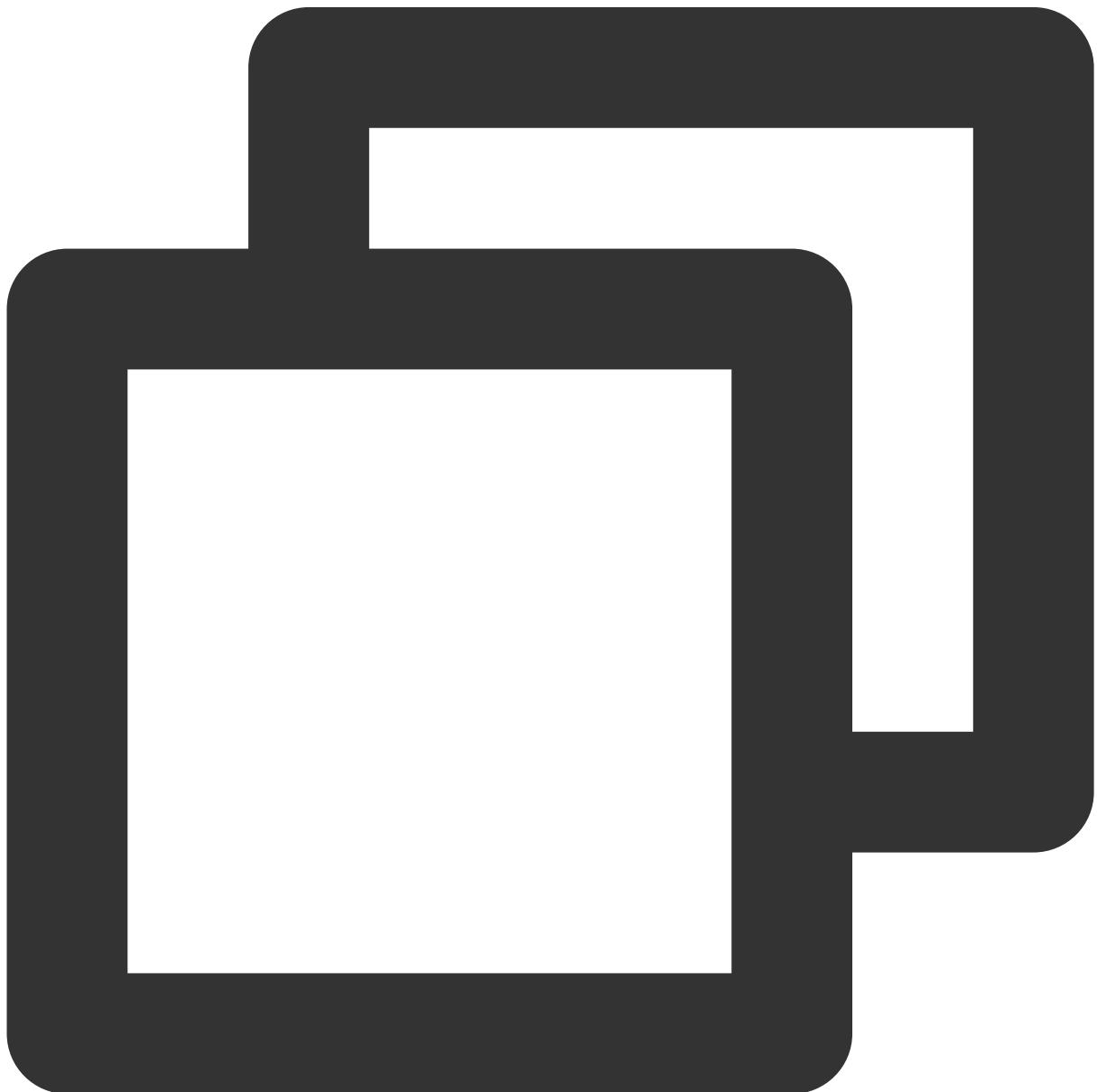
**Modifying** `log_slave_updates` **parameter**

In the source-replica or primary-replica reuse structure, if the `log-bin` parameter is enabled in the replica/secondary database, data operations directly performed in this database can be logged in the binlog, but data replications from the source/primary database to the replica/secondary database cannot be logged. Therefore, if the replica/secondary database is to be used as the source/primary database of another database, the `log_slave_updates` parameter needs to be enabled.

1. Log in to the source database.
2. Add the following content to the `my.cnf` configuration file of the source database.

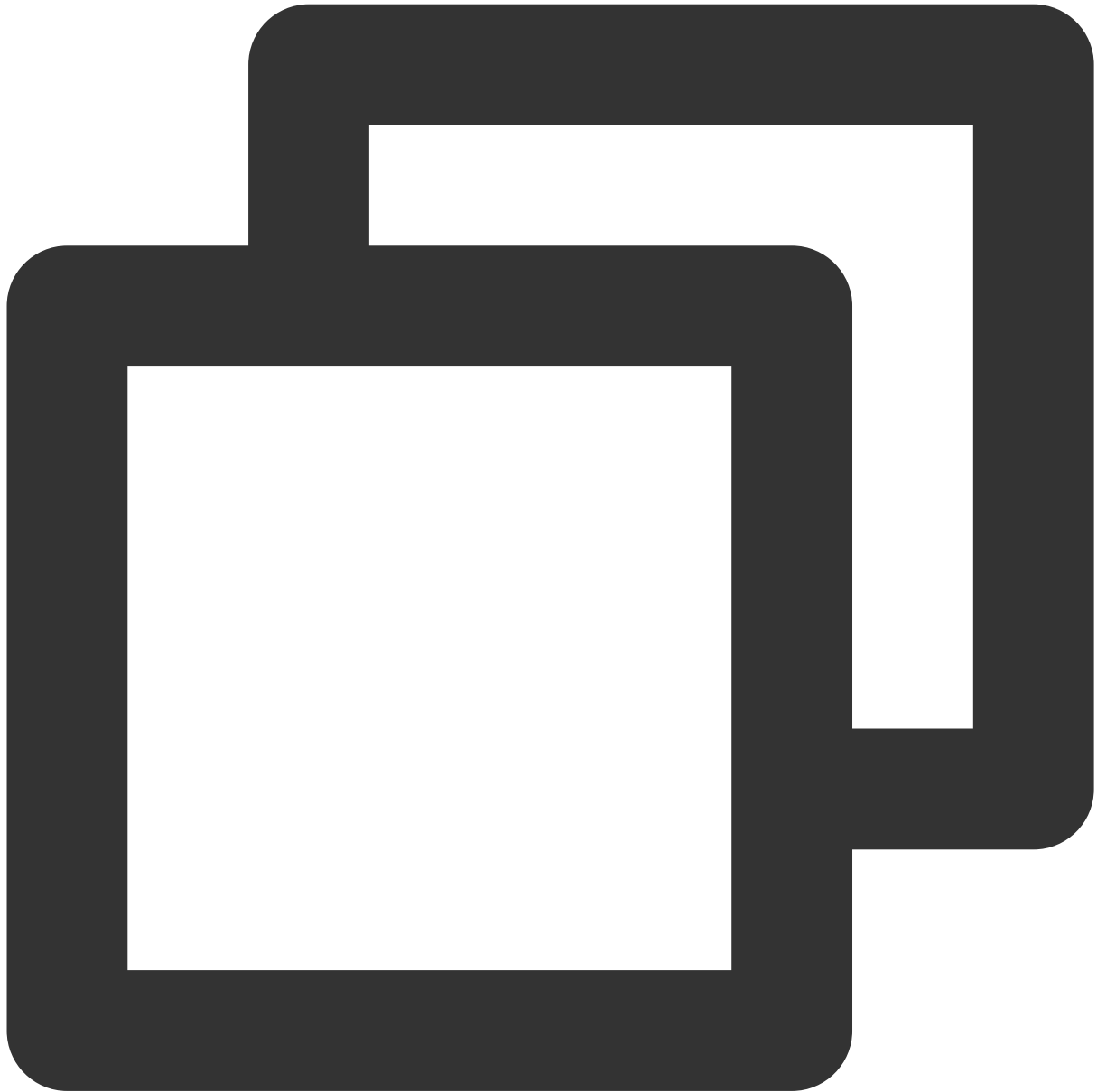
**Note**

The default path of the `my.cnf` configuration file is `/etc/my.cnf`, subject to the actual conditions.



```
log_slave_updates = ON
```

3. Run the following command to restart the source database:

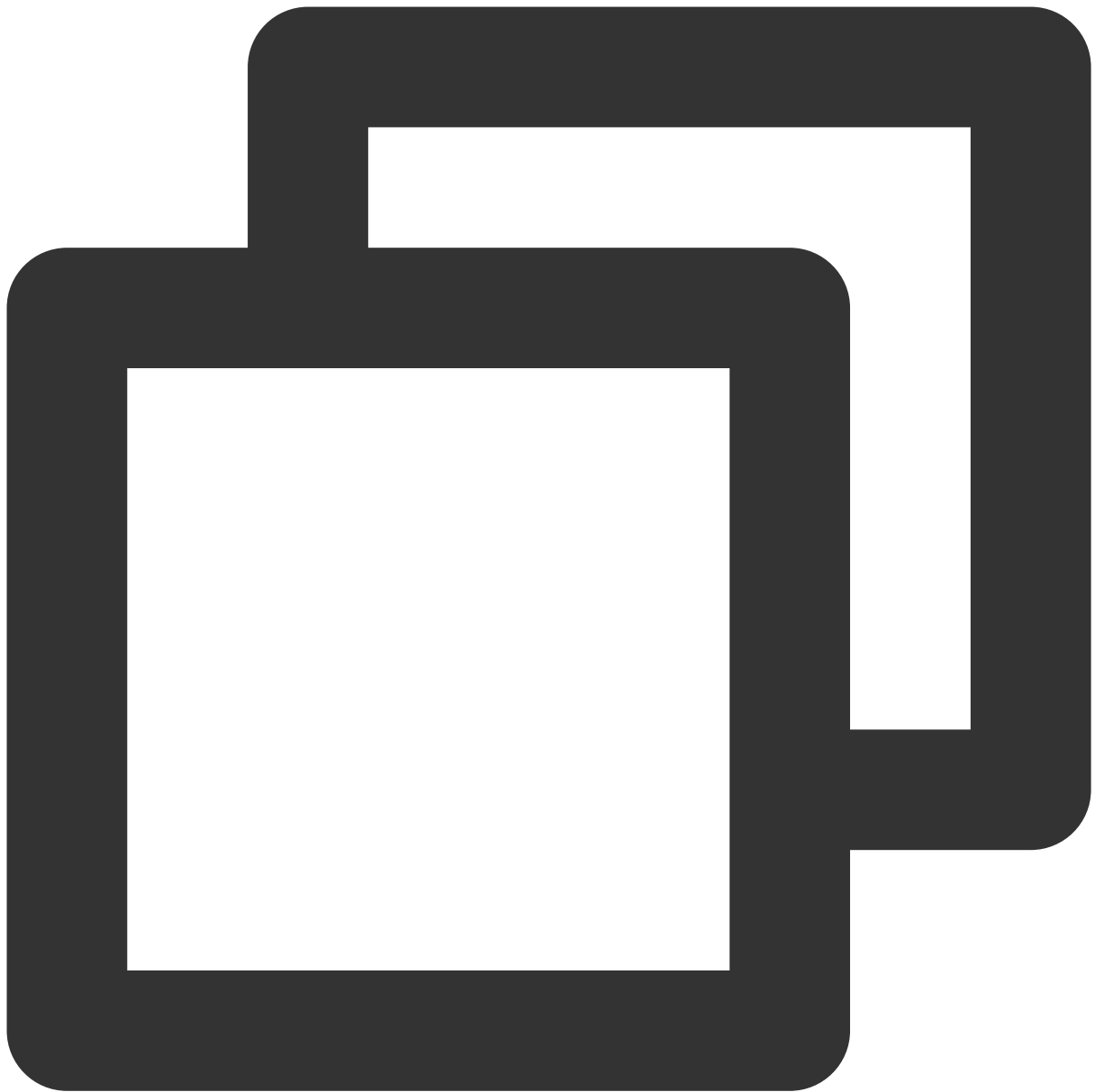


```
[$Mysql_Dir]/bin/mysqladmin -u root -p shutdown  
[$Mysql_Dir]/bin/safe_mysqld &
```

**Note**

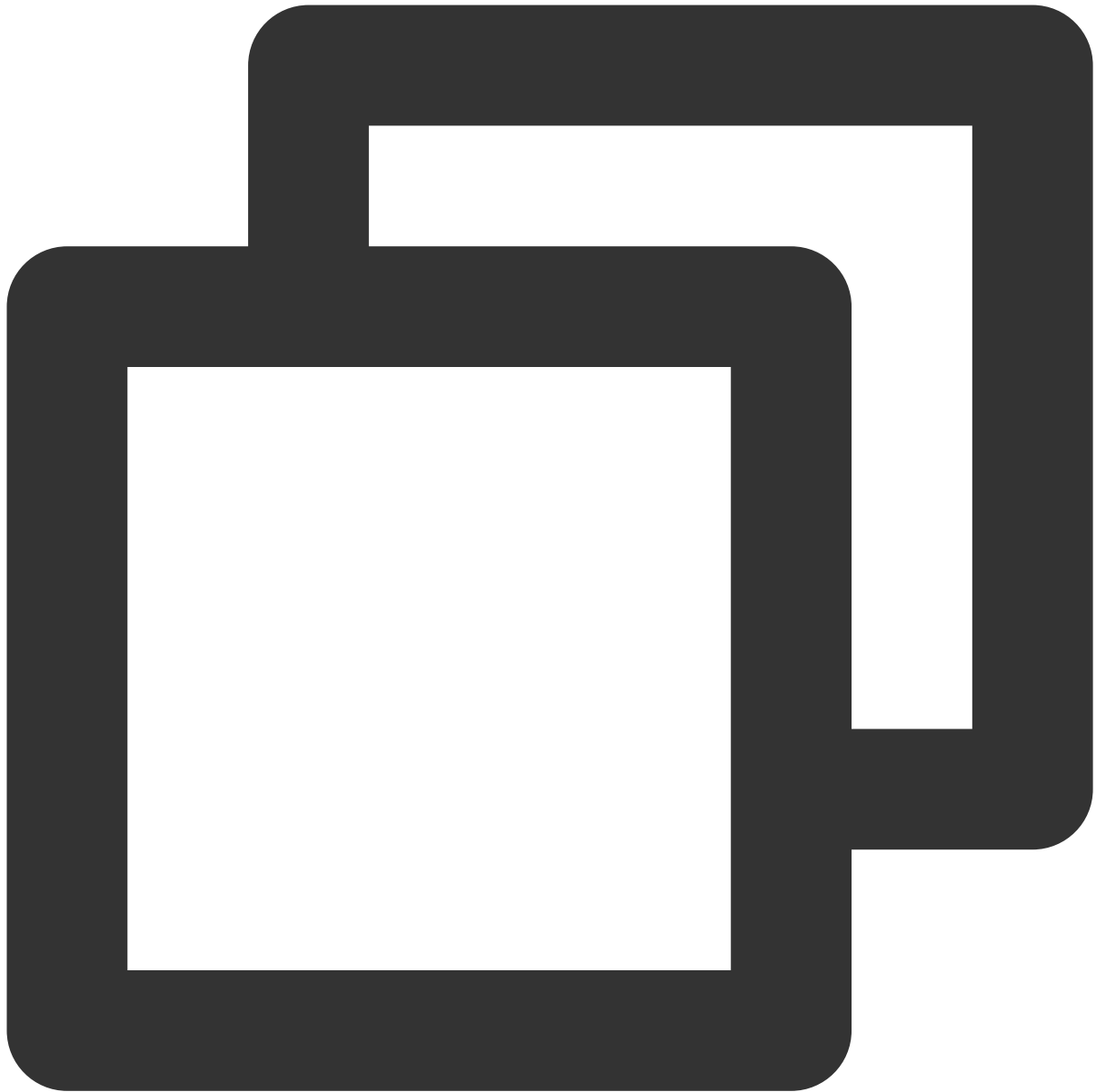
[ \$Mysql\_Dir ] is the installation path of the source database. Replace it with the actual path.

4. Check whether the configuration takes effect.



```
show variables like '%log_slave_updates%';
```

The system will display a result similar to the following:



```
mysql> show variables like '%log_slave_updates%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| log_slave_updates | ON    |
+-----+-----+
1 row in set (0.00 sec)
```

5. Run the verification task again.



# Data Subscription for TDSQL PostgreSQL

## Creating Data Subscription for TDSQL PostgreSQL Edition

Last updated : 2024-07-11 15:25:23

This scene introduces the instructions for using DTS to create a data subscription task for Tencent Cloud TDSQL PostgreSQL edition.

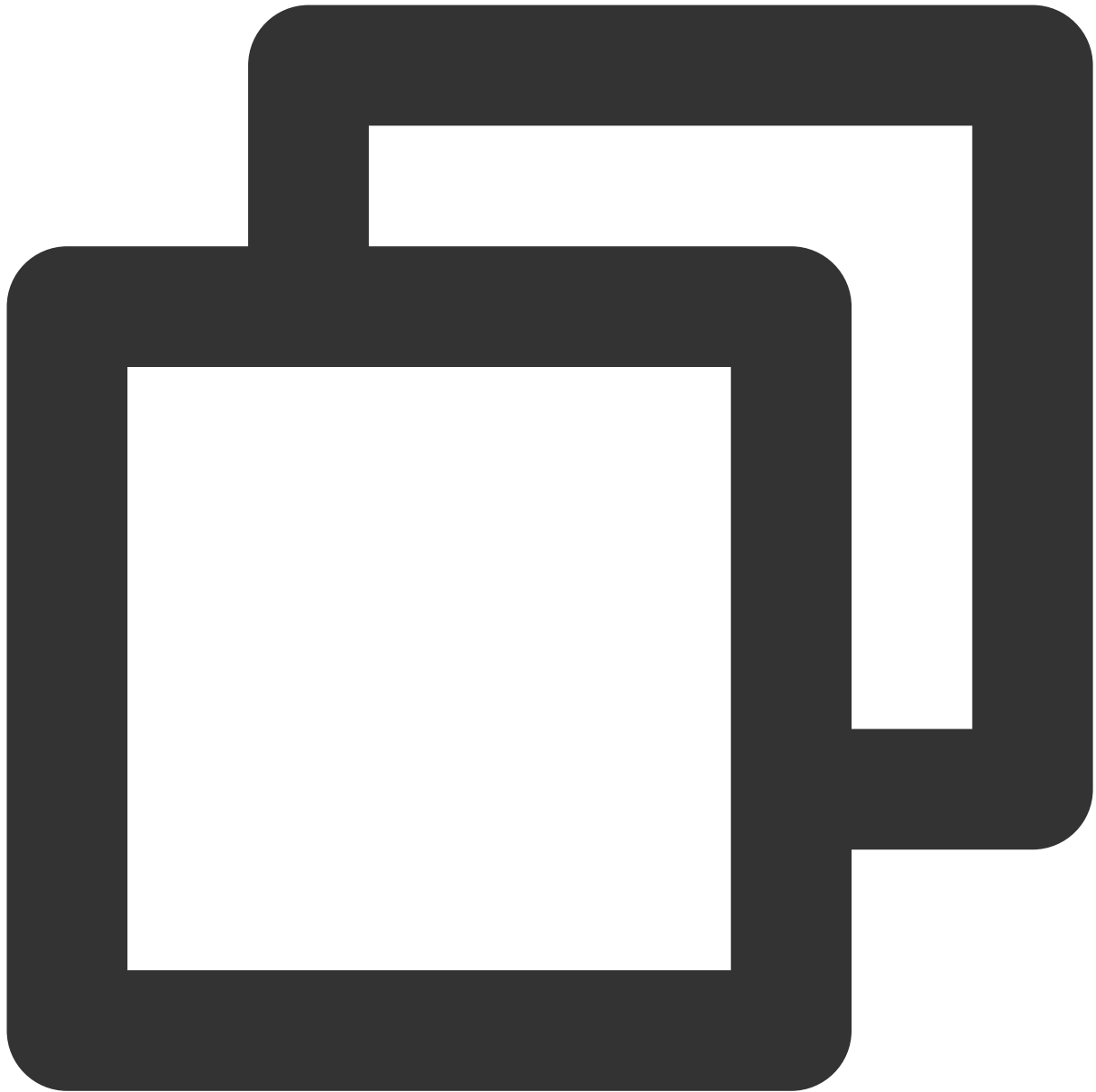
### Prerequisites

You have prepared TDSQL PostgreSQL edition to be subscribed to, and the database edition meets the requirements. Please see [Databases Supported by Data Subscription](#).

You have created a subscription account in the source instance, requiring the account to have the following permissions: LOGIN and REPLICATION.

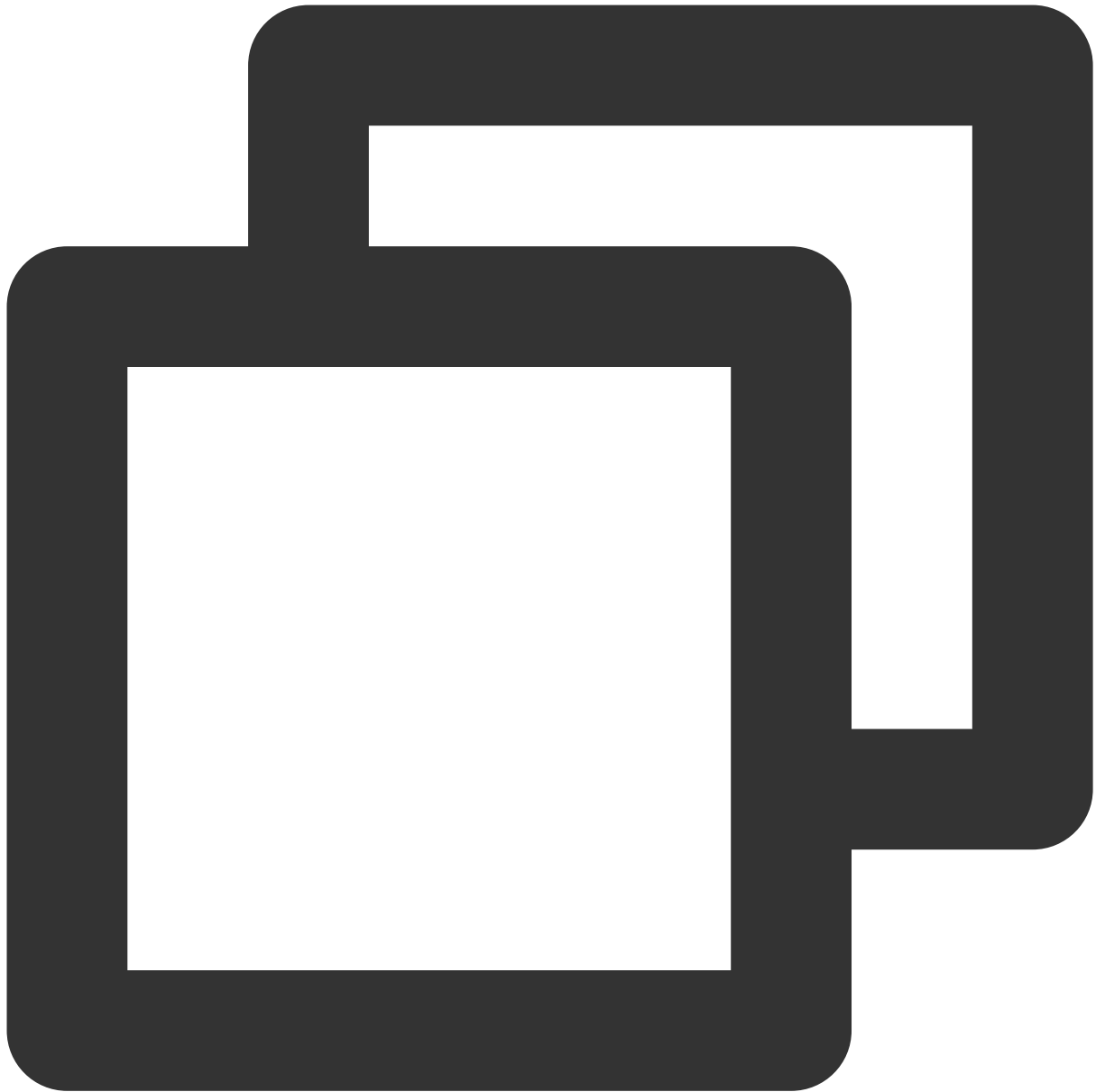
For LOGIN and REPLICATION authorization, please [submit a ticket](#).

The subscription account must have the select permission on the table to be subscribed to. For a complete database subscription, the subscription account must have the select permission for all tables under the schema. Specific authorization statements are as follows:



```
grant SELECT on all tables in schema "schema_name" to "migration account";
```

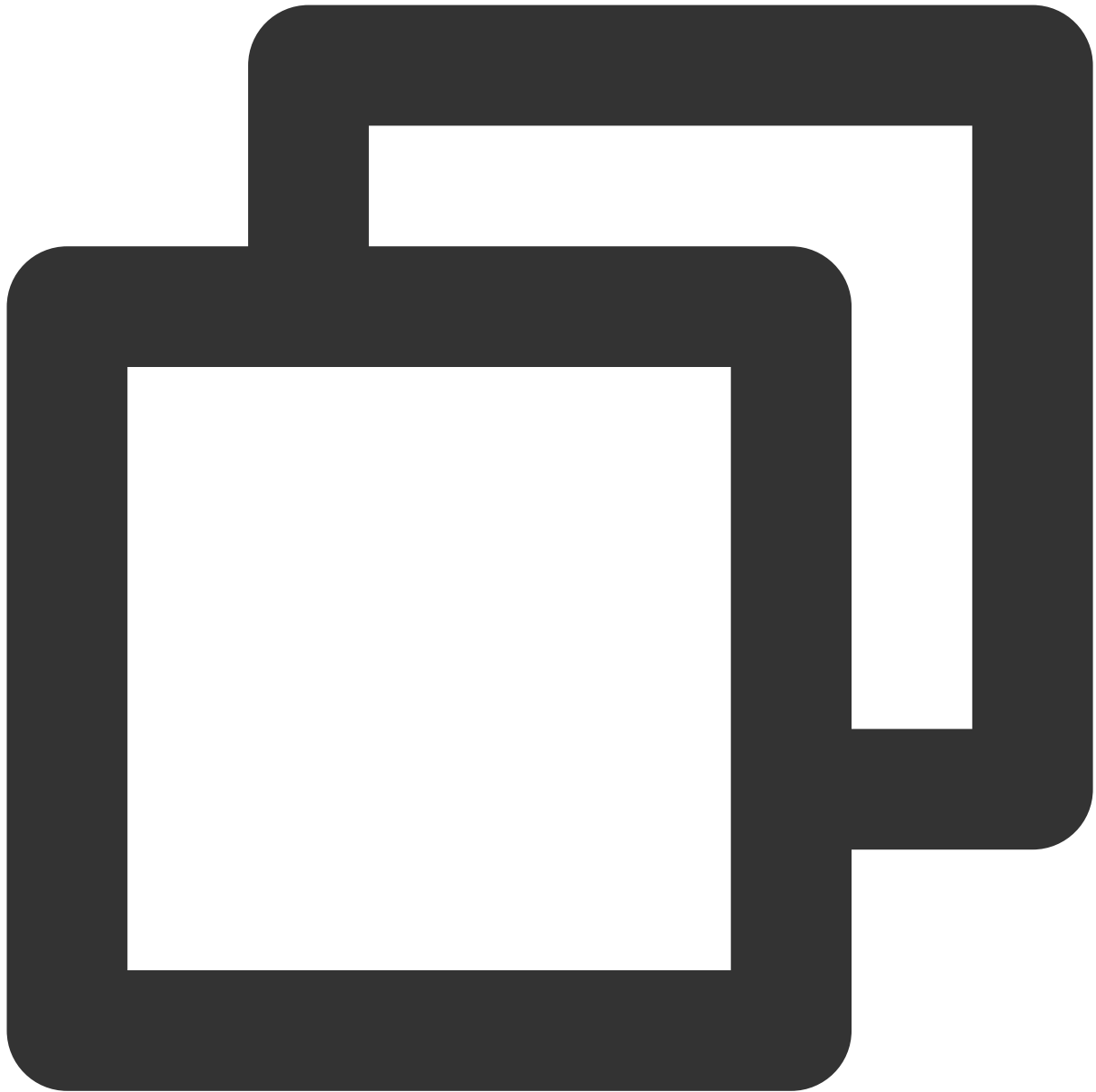
Users must have the SELECT permission on the pg\_catalog.pgxc\_node table. The specific authorization statement is as follows:



```
grant SELECT on pg_catalog.pgxc_node to "migration account";
```

The wal\_level of DN node must be logical.

If the table being subscribed to is a full replication table (the table creation statement contains the distribute by replication keyword), it must have a primary key; if the table being subscribed to is not a full replication table, it must have a primary key or have REPLICA IDENTITY as FULL; statement to modify the table's REPLICA IDENTITY to FULL:



```
alter table "table name" REPLICA IDENTITY FULL;
```

## Restrictions

Subscribed messages are stored in DTS built-in Kafka (single Topic), with a default retention time of 1 day as of now. A single Topic can have a maximum Storage capacity of 500 G. When the data Storage period exceeds 1 day, or the

data volume surpasses 500 G, the built-in Kafka will start to clear the earliest written data. Therefore, users should consume data promptly to avoid it being cleared before consumption.

The region of data consumption needs to match the region of the subscription task.

Currently not supporting data types related to gtsvector, pg\_dependencies, pg\_node\_tree, pg\_ndistinct, xml.

When the data subscription source is TDSQL PostgreSQL edition, direct execution of authorization statements is not supported. Therefore, permissions for the subscription account need to be configured through the [TDSQL Console](#) by clicking the instance ID, accessing the instance log-in information, and then authorizing the account via client log-in to the database.

During a subscription task, if operations such as modifying the Subscription object occur, it will cause the task to restart. After restarting, it might lead to duplicates when consuming data on the Kafka client.

DTS transmits data by the smallest data cell. Each marked checkpoint position represents a data cell. If a data cell has been completely transmitted when the task restarts, it will not cause data duplication; if a data cell is still being transmitted during a restart, it will need to be fetched again after the restart to ensure data integrity, leading to data duplication.

If users are concerned about duplicate data, please set up deduplication logic when consuming data.

## SQL Operations for Subscription Supported

Operation Type	Supported SQL Operations
DML	INSERT, UPDATE, DELETE

## Directions

1. Log in to [DTS Console](#), choose **Data Subscription** in the left sidebar, and click **Create Subscription**.
2. On the page for creating a new Data Subscription, select the appropriate configuration and click **Buy Now**.

Billing Mode: Supports **Monthly Subscription** and **Pay as you go**.

Region: The region must be the same as that of the database instance to be subscribed to.

Database: Select your actual database type.

Edition: Select **Kafka Edition**, which supports direct consumption through a Kafka client.

Subscription Instance Name: Edit the current name of data subscription instance.

3. After a successful purchase, return to the data subscription list, click **Operation** column's **Configure Subscription** to configure the newly purchased subscription. You can use it only after the configuration is complete.
4. On the **Configure data subscription** page, select the appropriate configuration and click **Next**.

Instance: Select the corresponding database instance. Currently, read-only and disaster recovery instances do not support data subscription.

Database Account: Add the account and password of the subscription instance, the LOGIN and REPLICATION permissions of the account and the SELECT permissions for all objects and the pg\_catalog.pgxc\_node table.

5. On the Subscription Type and Object Selection page, choose the type of subscription and click **Save**.

The subscription type is Data Update (subscribes to data updates for selected objects, including INSERT, UPDATE, DELETE operations).

Kafka partition strategy: Support partitioning by table names.

6. On the pre-verification page, the pre-verification task is expected to run for 2–3 minutes. After passing pre-verification, click **Start** to complete the data subscription task configuration.

**Note:**

If verification fails, modify the task in the instance to be subscribed to as prompted and initiate the verification again.

7. After clicking Start, the subscription task will initialize, expected to take 3–4 minutes. Once initialization is successful, it enters the **Running** status.

8. [Creating Consumer Group](#), the Kafka edition of data subscription supports creating multiple consumer groups for multi-point consumption. Consumption in the Kafka edition relies on Kafka's consumer groups, hence it is necessary to create a consumer group before data consumption.

9. After the subscription instance enters the Running status, data consumption can begin. Consumption in Kafka requires password authentication, for specific examples, please see Data Consumption Demo. We provide Demo code in multiple languages and explain the main process of consumption and key data structures.

# Consuming TDSQL for PostgreSQL Data

Last updated : 2023-11-21 20:17:30

## Overview

In data subscription (Kafka Edition, where the current Kafka server version is v2.6.0), you can consume the subscribed data through Kafka 0.11 or later available at [DOWNLOAD](#). This document provides client consumption demos for you to quickly test the process of data consumption and understand the method of data parsing.

## Note

**The demo only prints out the consumed data and contains no instructions. You need to write your own data processing logic based on the demo.** You can also use Kafka clients in other languages to consume and parse data.

Currently, data subscription to Kafka for consumption can be implemented over the Tencent Cloud private network but not the public network. In addition, the subscribed database instance and the data consumer must be in the same region.

In scenarios where only the specified databases and tables (part of the source instance objects) are subscribed and single-partition Kafka topics are used, only the data of the subscribed objects will be written to Kafka topics after DTS parses the incremental data. The data of non-subscription objects will be converted into empty transactions and then written to Kafka topics. Therefore, there are empty transactions during message consumption. The BEGIN and COMMIT messages in the empty transactions contain the GTID information, which can ensure the GTID continuity and integrity.

To ensure that data can be rewritten from where the task is paused, DTS adopts the checkpoint mechanism for data subscription. Specifically, when messages are written to Kafka topics, a checkpoint message is inserted every 10 seconds to mark the data sync offset. When the task is resumed after being interrupted, data can be rewritten from the checkpoint message. The consumer commits a consumption offset every time it encounters a checkpoint message so that the consumption offset can be updated timely.

## Downloading Consumption Demos

Currently, TDSQL for PostgreSQL data subscription only supports the Protobuf format. The following demos already contain the ProtoBuf protocol file, so you don't need to download it separately. If you [download](#) it on your own, use Protobuf 3.X for code generation to ensure that data structures are compatible.

The logic description of the consumption demo for TDSQL for PostgreSQL is similar to that for MySQL. For more information, see [Demo Description](#).

Demo Language	Protobuf Demo Download
Go	<a href="#">Address</a>
Java	<a href="#">Address</a>
Python	<a href="#">Address</a>

## Directions for the Java Demo

Compiling environment: The package management tool Maven or Gradle, and JDK8. You can choose a desired package management tool. The following takes Maven as an example.

Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install JRE 8.

Follow the steps below:

1. Create a data subscription task (NewDTS) as instructed in [Creating TDSQL for PostgreSQL Data Subscription](#).
2. Create one or multiple consumer groups as instructed in [Adding Consumer Group](#).
3. Download the Java demo and decompress it.
4. Access the decompressed directory. Maven model and pom.xml files are placed in the directory for your convenience.

Package with Maven by running `mvn clean package`.

5. Run the demo.

After packaging the project with Maven, go to the target folder `target` and run the following code: `java -jar consumerDemo-avro-1.0-SNAPSHOT.jar --brokers xxx --topic xxx --group xxx --user xxx --password xxx --trans2sql`

`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement. In Java code, if this parameter is carried, the conversion will be enabled.

### Note

If `trans2sql` is carried, `javax.xml.bind.DatatypeConverter.printHexBinary()` will be used to convert byte values to hex values. You should use JDK 1.8 or later to avoid incompatibility. If you don't need SQL conversion, comment this parameter out.

6. Observe the consumption.



```
BEGIN
-->[partition: 0, offset: 87008, partitionSeq: 87009] [mysql-bin.000004:24574], happenedAt: 2021-03-01T
17:49:14
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 61)
-->[partition: 0, offset: 87008, partitionSeq: 87009] [mysql-bin.000004:24605], happenedAt: 2021-03-01T
17:49:14
COMMIT
```

## Directions for the Golang Demo

Compiling environment: Go 1.12 or later, with the Go module environment configured.

Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance).

Follow the steps below:

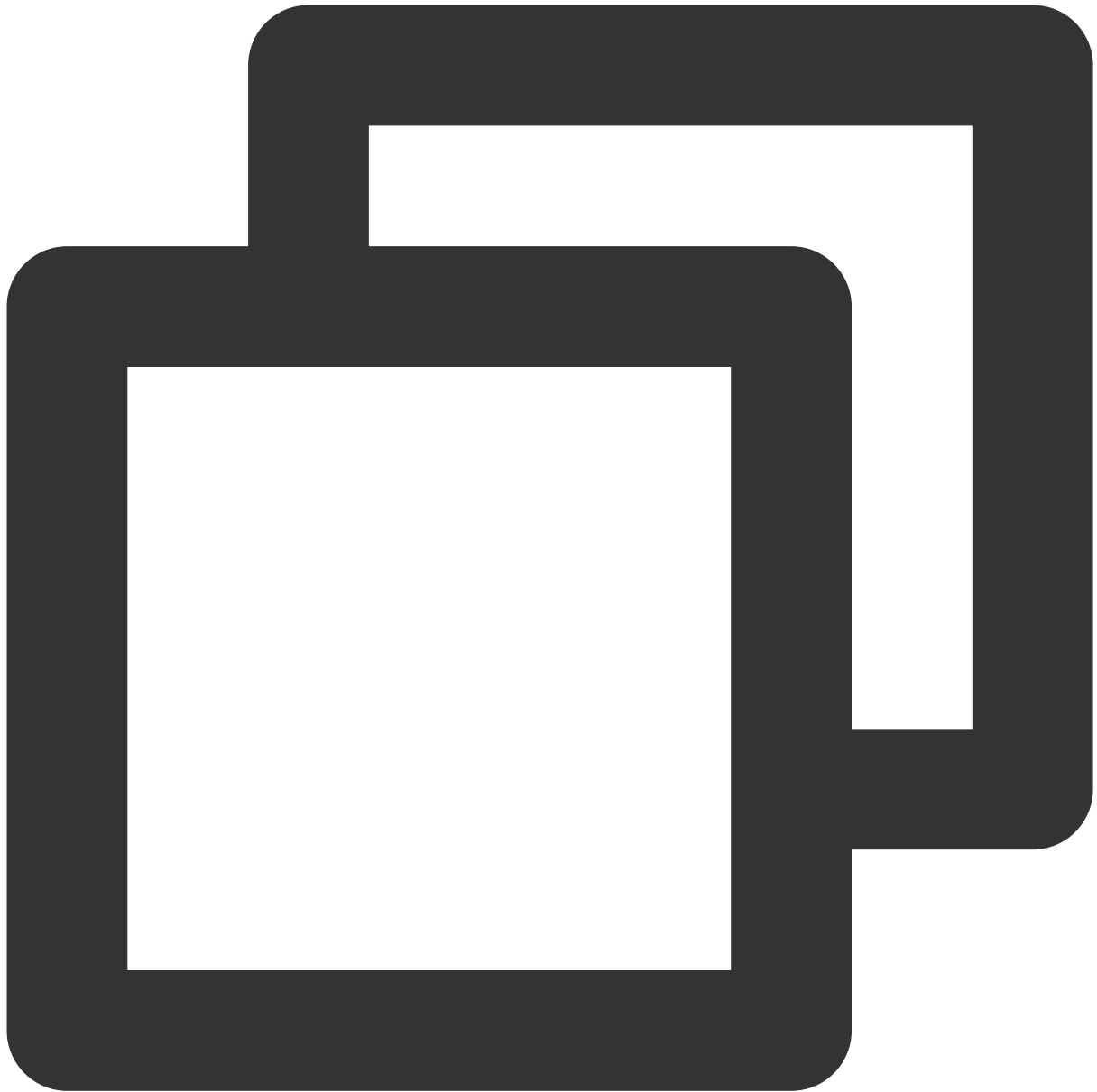
1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).
2. Create one or multiple consumer groups as instructed in [Adding Consumer Group](#).
3. Download the Go demo and decompress it.
4. Access the decompressed directory and run `go build -o subscribe ./main/main.go` to generate the executable file `subscribe`.
5. Run `./subscribe --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=true`.  
`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).  
`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).  
`trans2sql` indicates whether to enable conversion to SQL statement.
6. Observe the consumption.

```
BEGIN
-->[partition: 0, offset: 86991, partitionSeq: 86992] [mysql-bin.000004:24272], happenedAt: 2021-03-01
17:47:49 +0800 CST
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 60)
-->[partition: 0, offset: 86991, partitionSeq: 86992] [mysql-bin.000004:24303], happenedAt: 2021-03-01
17:47:49 +0800 CST
COMMIT
```

## Directions for the Python3 Demo

Compiling and runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install Python 3 and pip3 (for dependency package installation).

Use `pip3` to install the dependency package:



```
pip install flag
pip install kafka-python
pip install avro
```

Follow the steps below:

1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).
2. Create one or multiple consumer groups as instructed in [Adding Consumer Group](#).
3. Download Python3 demo and decompress it.

4. Run `python main.py --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=1` .

`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group` , `user` , and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement.

5. Observe the consumption.

```
BEGIN
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24876], happenedAt: 2021-03-01
20:43:31
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 62)
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24907], happenedAt: 2021-03-01
20:43:31
COMMIT
```

# Fix for Verification Failure

## Database Connection Check

Last updated : 2023-08-25 16:51:50

### Check Details

The source and target databases need to be normally connected, and if not, a connection failure will be reported.

### Causes

The network or server where the source database resides has a security group or firewall configured. For more information, see [Failed Connectivity Test > Security Group or Firewall Configured in Network or Server of Source Database](#).

The source IP addresses are blocked by the source database. For more information, see [Failed Connectivity Test > Source IP Addresses Blocked in Source Database](#).

The network port is closed. For more information, see [Failed Connectivity Test > Closed Network Port](#).

The database account or password is incorrect.

### Troubleshooting

Refer to the causes above based on the actual scenario and troubleshoot as instructed.

# Version Check

Last updated : 2023-08-25 16:53:12

## Check Details

Log in to the database, run `select tbase_version();`, and `Tbase_V2.xx` must be returned.

## Troubleshooting

Select the ID of the TDSQL for PostgreSQL instance on the supported version.

# Peripheral Check

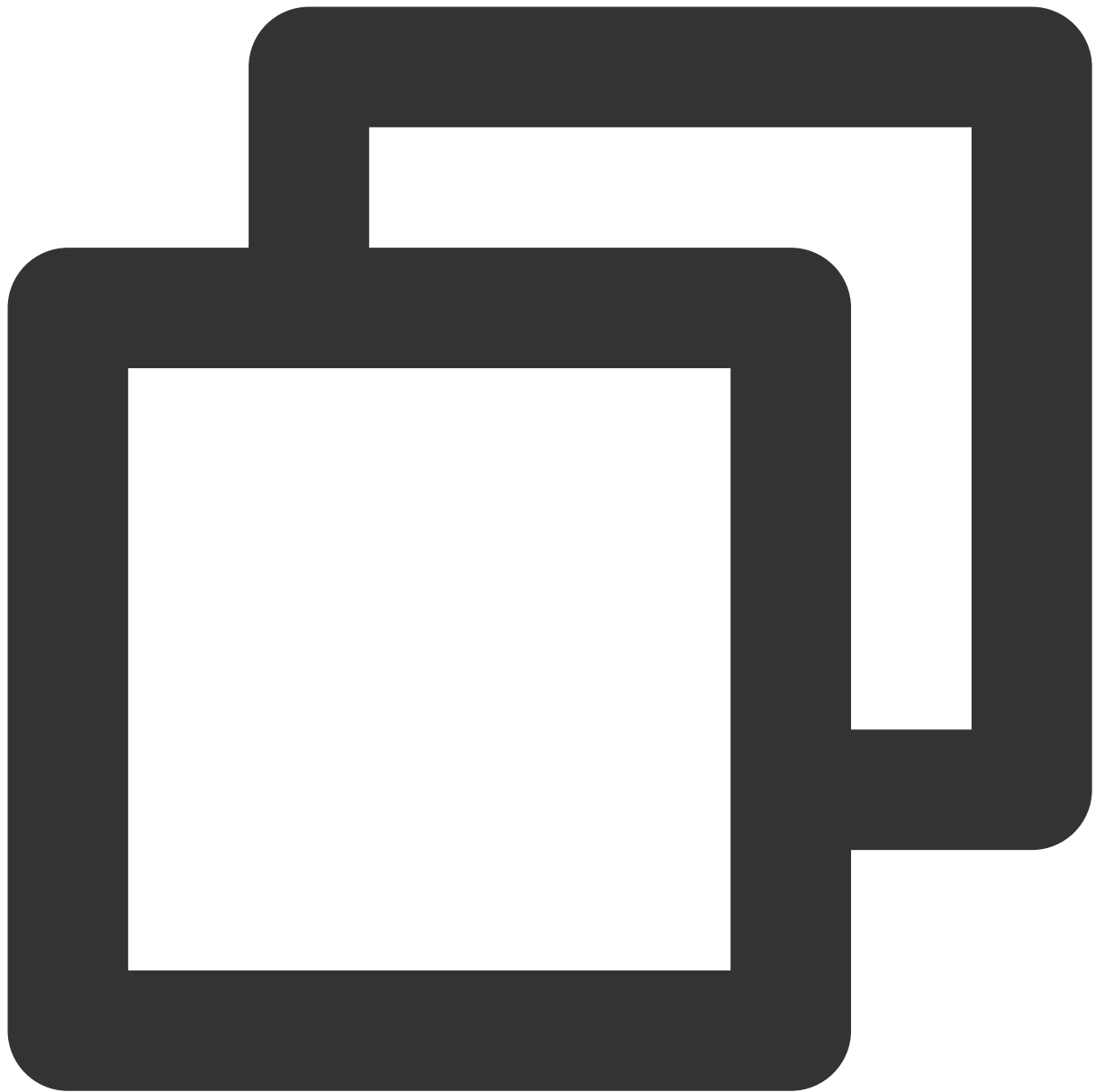
Last updated : 2023-08-25 16:45:59

## Check Details

The `wal_level` of the DN node must be `logical` .

## Troubleshooting

1. Log in to the source database.
2. Open the `postgresql.conf` file and modify `wal_level` .



```
`wal_level` = `logical`
```

3. After the modification is completed, restart the database.

# Source Instance Permission Check

Last updated : 2023-07-19 17:04:58

## Check Details

You must have the REPLICATION permission, that is, `pg_roles.rolreplication` .

You must have the SELECT permission of the subscribed table. For entire database subscription, you must have the SELECT permission of all the tables under the schema.

For more information, see [Creating TDSQL for PostgreSQL Data Subscription](#).

## Troubleshooting

If you don't have the operation permissions, get authorized based on the permission requirements in the check details and run the verification task again.



# Constraint Check

Last updated : 2023-08-25 16:46:52

## Check Details

The subscribed table must have a primary key or have REPLICA IDENTITY as FULL.

## Troubleshooting

If the verification fails, modify the corresponding table.

# MongoDB Data Subscription

## Creating Data Subscription for MongoDB

Last updated : 2024-07-11 15:22:52

This scenario describes how to create a data subscription task in DTS for TencentDB for MongoDB.

### Version Description

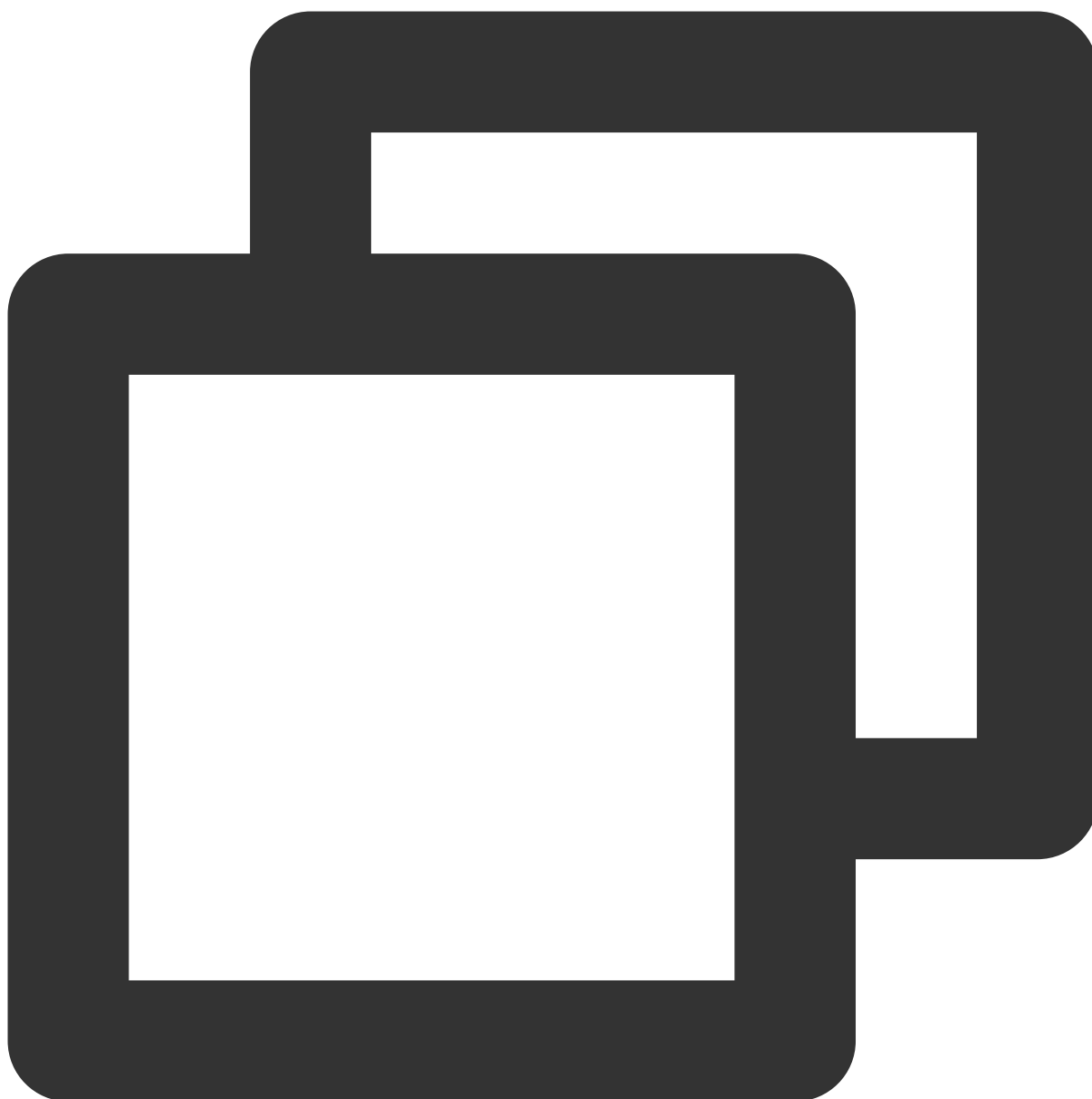
Currently, data subscription is supported only for TencentDB for MongoDB 3.6, 4.0, 4.2, and 4.4.

TencentDB for MongoDB 3.6 only supports collection-level subscription.

### Prerequisites

You have prepared a TencentDB instance to be subscribed to, and the database version meets the requirements. For more information, see [Databases Supported by Data Subscription](#).

We recommend you create a read-only account in the source instance by seeing the following syntax. You can also do this in the TencentDB for MongoDB console.



```
# Create an Instance-Level Read-Only Account
use admin
db.createUser({
  user: "username",
  pwd: "password",
  roles:[
    {role: "readAnyDatabase",db: "admin"}
  ]
})

# Create a Database-Specific Read-Only Account
```

```
use admin
db.createUser({
  user: "username",
  pwd: "password",
  roles:[
    {role: "read",db: "Name of the specified database"}
  ]
})
```

## Notes

Subscribed messages are stored in DTS built-in Kafka (single Topic), with a default retention time of 1 day as of now. A single Topic can have a maximum Storage capacity of 500 GB. When the data Storage period exceeds 1 day, or the data volume surpasses 500 GB, the built-in Kafka will start to clear the earliest written data. Therefore, users should consume data promptly to avoid it being cleared before consumption.

The region of data consumption needs to match the region of the subscription task.

The Kafka built in DTS has a certain upper limit for processing individual messages. When a single row of data in the source database exceeds 10 MB, this row may be discarded in the consumer.

After the database or collection specified for the selected subscription object is deleted from the source database, the subscription data (change stream) of the database or collection will be invalidated. Even if the database or collection is rebuilt in the source database, the subscription data cannot be resubscribed. In this case, you need to reset the subscription task and select the subscription object again.

## SQL Operations for Subscription Supported

Operation Type	Supported SQL Operations
DML	INSERT,UPDATE,DELETE
DDL	INDEX:createIndexes,createIndex,dropIndex,dropIndexes COLLECTION:createCollection,drop,collMod,renameCollection DATABASE:dropDatabase,copyDatabase

## Subscription Configuration Steps

1. Log in to [DTS Console](#), choose **Data Subscription** in the left sidebar, and click **Create Subscription**.

2. On the page for Creating Data Subscription, select the appropriate configuration and click **Buy Now**.

Billing Model: Supports **Monthly subscription** and **Pay as you go**.

Region: The region must be the same as that of the database instance to be subscribed to.

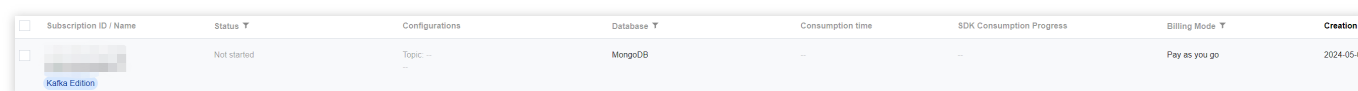
Database: Choose **MongoDB**.

Edition: Select **Kafka Edition**, which supports direct consumption through a Kafka client.

Subscribed Instance Name: Edit the name of the current data subscription instance.

Quantity: You can purchase up to 10 tasks at a time.

3. After successful purchase, return to the data subscription list, select the task just purchased, and click **Configure Subscription** in the **Operation** column.



<input type="checkbox"/>	Subscription ID / Name	Status ▾	Configurations	Database ▾	Consumption time	SDK Consumption Progress	Billing Mode ▾	Creation Time
<input type="checkbox"/>	[Redacted]	Not started	Topic: -- --	MongoDB	--	--	Pay as you go	2024-05-01

4. On the configure data subscription page, after configuring the source database information, click **Test Connectivity**, after passing, click **Next**.

Access Type: Currently, only supports **Database**.

Instance Name: Select the TencentDB instance ID.

Database Account/Password: Add the username and password for the subscription instance. The account has read-only permissions.

Number of Kafka Partitions: Set the number of Kafka partitions. Increasing the number can improve the speed of data write and consumption. A single partition can guarantee the order of messages, while multiple partitions cannot. If you have strict requirements for the order of messages during consumption, set this value to 1.

1 Select Instance
2 Subscription Type and Object
3 Pre-verification

### Data Subscription Task Setting

Subscription ID / Name subs-8286u88yg0 (name-8286u88yg0)

Instance Type MongoDB

Region South China(Guangzhou)

Access Type \* Database [Access Type Description](#)

Instance Name \* Please select

Account \* Please enter the account

Password \* Please enter password

Test Connectivity

Note: You are using the data subscription service.  
For data security, please read [Data Subscription](#) carefully before creating a data subscription task.

Next

5. In the Subscription Type and Object Selection Page, after selecting the subscription parameters, click **Save**.

Parameter	Description
Data Subscription Type	It is Change Stream by default and cannot be modified.
Subscription Object Level	Subscription level includes all instances, library, and collection. All Instances: Subscribe to the data of all instances. Library: Subscribe to the library-level data. After selection, only one library can be chosen in the task settings below. Collection: Subscribe to the collection-level data. After selection, only one collection can be chosen in the task settings below.
Task Configuration	Select the database or collection to be subscribed to. You can select only one database or collection.
Output Aggregation Settings	If this option is selected, the execution order of the aggregation pipeline is determined by the configuration order on the page. For more information and examples of the aggregation pipeline, see <a href="#">MongoDB Official Documentation</a> .
Kafka Partitioning Policy	Partitioning by Collection Name: Partitions the subscribed data from the source database by collection name. Once set, data with the same collection name will be written to the same Kafka partition. Custom Partitioning Policy: Database and collection names of the subscribed data are matched through a regex first. Then, matched data is partitioned by collection name or collection name + <code>objectid</code> .

6. On the pre-verification page, the pre-verification task is expected to run for 2–3 minutes. After passing pre-verification, click **Start** to complete the data subscription task configuration.

**Note:**

If verification fails, correct it according to the [Handling Methods for Validation Failure](#) and initiate the verification again.

7. The subscription task will be initialized, expected to run for 3–4 minutes. After successful initialization, it will enter the **Running** status.

## Subsequent Operations

### 1. [Create Consumer Group](#).

The consumption in data subscription (Kafka Edition) relies on Kafka's consumer groups, so you need to create a consumer group before consuming data. Data subscription (Kafka Edition) supports the creation of multiple consumer groups for multi-point consumption.

### 2. [Consume Subscription Data](#).

After the subscription task enters the running state, you can start consuming data. Consumption with Kafka requires password authentication. For details, see Demo in [Consume Subscription Data](#). We provide demo codes in various languages and have explained the main process of consumption and the key data structures.

# Consuming MongoDB Data

Last updated : 2023-11-21 20:19:35

## Overview

In data subscription (Kafka Edition, where the current Kafka server version is v2.6.0), you can consume the subscribed data through Kafka 0.11 or later available at [DOWNLOAD](#). This document provides client consumption demos for you to quickly test the process of data consumption and understand the method of data parsing.

## Note

**The demo only prints out the consumed data and does not contain any usage instructions. You need to write your own data processing logic based on the demo.** You can also use Kafka clients in other languages to consume and parse data.

Currently, data subscription to Kafka for consumption can be implemented over the Tencent Cloud private network but not the public network. In addition, the subscribed database instance and the data consumer must be in the same region.

The Kafka built in DTS data subscription has an upper limit for processing a single message. When a single row of data in the source database exceeds 5 MB, the subscription task may report an error.

To ensure that data can be rewritten from where the task is paused, DTS adopts the checkpoint mechanism for data subscription. Specifically, when messages are written to Kafka topics, a checkpoint message is inserted every 10 seconds to mark the data sync offset. When the task is resumed after being interrupted, data can be rewritten from the checkpointed offset. The consumer commits a consumption offset every time it encounters a checkpoint message so that the consumption offset can be updated timely.

## Downloading Consumption Demos

Currently, MongoDB data subscription only supports the JSON format. The following demos already contain the JSON protocol file, so you don't need to download it separately.

The consumption demo uses the native changeStream format. For details, see the [MongoDB Manual](#).

Demo Language	JSON Demo Download
Go	<a href="#">Address</a>
Java	<a href="#">Address</a>



Python

Address

## Directions for the Java Demo

Compiling environment: Maven and JDK8. You can choose a desired package management tool. The following takes Maven as an example.

Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install JRE 8.

Follow the steps below:

1. Create a data subscription task (NewDTS) as instructed in [Creating Data Subscription to TencentDB for MariaDB](#).
2. Create one or multiple consumer groups as instructed in [Creating Consumer Group](#).
3. Download the Java demo and decompress it.
4. Access the decompressed directory. Maven model and pom.xml files are placed in the directory for your use as needed.

Package with Maven by running `mvn clean package`.

5. Run the demo.

After packaging the project with Maven, go to the target folder `target` and run the following code:

```
java -jar consumerDemo-json-1.0-SNAPSHOT.jar --brokers xxx --topic xxx --group xxx  
--user xxx --password xxx --trans2sql --trans2canal
```

`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement. In Java code, if this parameter is carried, the conversion will be enabled.

`trans2canal` indicates whether to print the data in Canal format. If this parameter is carried, the conversion will be enabled. Currently, this parameter is only used for data in JSON format.

6. Observe the consumption.

```
BEGIN  
-->[partition: 0, offset: 87008, partitionSeq: 87009] [mysql-bin.000004:24574], happenedAt: 2021-03-01T  
17:49:14  
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 61)  
-->[partition: 0, offset: 87008, partitionSeq: 87009] [mysql-bin.000004:24605], happenedAt: 2021-03-01T  
17:49:14  
COMMIT
```

## Directions for the Golang Demo

Compiling environment: Go 1.12 or later, with the Go module environment configured.

Runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance).

Follow the steps below:

1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).

2. Create one or multiple consumer groups as instructed in [Creating Consumer Group](#).

3. Download the Go demo and decompress it.

4. Access the decompressed directory and run `go build -o subscribe ./main/main.go` to generate the executable file `subscribe`.

5. Run `./subscribe --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=true`.

`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group`, `user`, and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement.

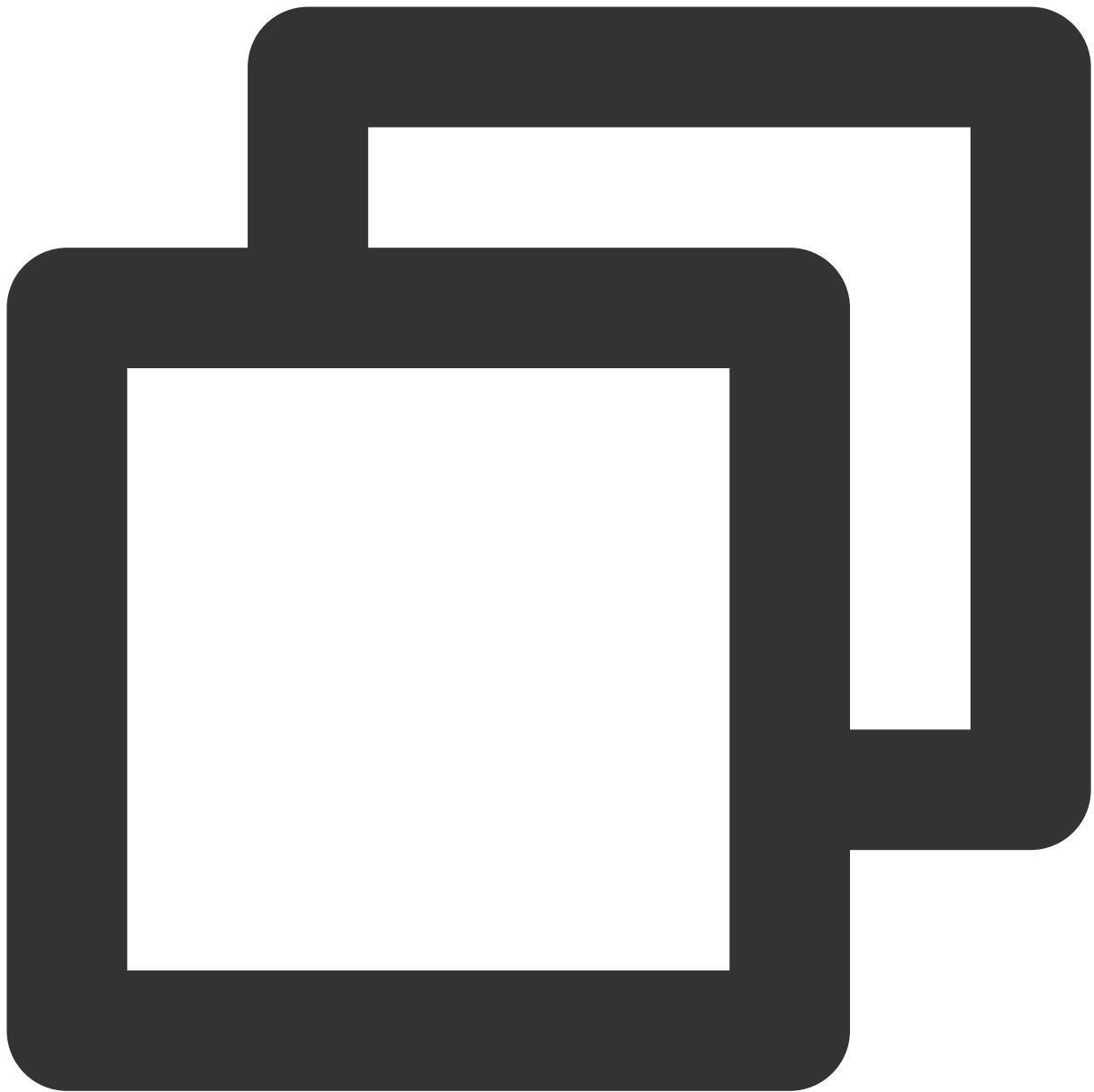
6. Observe the consumption.

```
BEGIN
-->[partition: 0, offset: 86991, partitionSeq: 86992] [mysql-bin.000004:24272], happenedAt: 2021-03-01
17:47:49 +0800 CST
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 60)
-->[partition: 0, offset: 86991, partitionSeq: 86992] [mysql-bin.000004:24303], happenedAt: 2021-03-01
17:47:49 +0800 CST
COMMIT
```

## Directions for the Python3 Demo

Compiling and runtime environment: Tencent Cloud CVM (which can access the private network address of the Kafka server only if it is in the same region as the subscribed instance). Install Python 3 and pip3 (for dependency package installation).

Use `pip3` to install the dependency package:



```
pip install flag
pip install kafka-python
```

Follow the steps below:

1. Create a data subscription task (NewDTS) as instructed in [Creating MySQL or TDSQL for MySQL Data Subscription](#).
2. Create one or multiple consumer groups as instructed in [Creating Consumer Group](#).
3. Download Python3 demo and decompress it.

4. Run `python main.py --brokers=xxx --topic=xxx --group=xxx --user=xxx --password=xxx --trans2sql=1` .

`brokers` is the private network access address for data subscription to Kafka, and `topic` is the subscription topic, which can be viewed on the **Subscription details** page as instructed in [Viewing Subscription Details](#).

`group` , `user` , and `password` are the name, account, and password of the consumer group, which can be viewed on the **Consumption Management** page as instructed in [Managing Consumer Group](#).

`trans2sql` indicates whether to enable conversion to SQL statement.

5. Observe the consumption.

```
BEGIN
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24876], happenedAt: 2021-03-01
20:43:31
INSERT INTO `kafka-subscribe`.`table1` VALUES (_binary'subscribe-kafka', 62)
-->[partition: 0, offset: 89083, partitionSeq: 89084] [mysql-bin.000004:24907], happenedAt: 2021-03-01
20:43:31
COMMIT
```

# Task Management

## Task Status Description

Last updated : 2024-07-08 17:25:39

Status	Description
Not started	The purchase has been completed, but no subscription task has been configured.
Checking	The subscription task is being checked.
Verification passed	The subscription task passed the verification.
Verification failed	The subscription task failed the verification.
Enabling	The subscription task is ready to run.
Task running	The subscription task is running.
Stopping	The subscription task is being stopped when a subscription object is reset.
Task error	The subscription task is interrupted due to an error.

# Viewing Subscription Details

Last updated : 2021-12-24 18:26:13

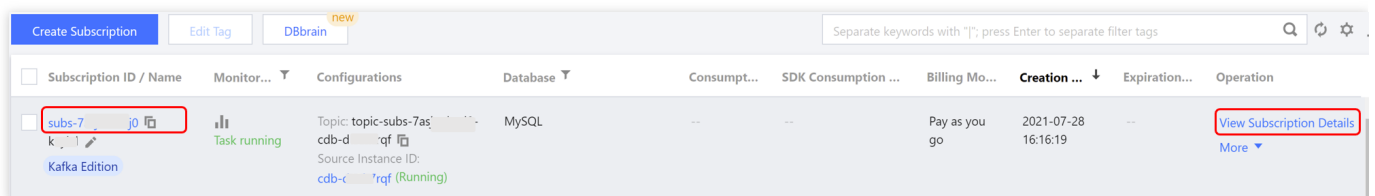
After successfully creating a data subscription task, you can view its details.

## Prerequisites

You have [created a data subscription task](#).

## Directions

1. Log in to the [DTS console](#) and select **Data Subscription** on the left sidebar to enter the data subscription page.
  - Method 1: select the target subscription task and click its name.
  - Method 2: select the target subscription task and click **View Subscription Details** in the **Operation** column.



Create Subscription Edit Tag DBbrain new Separate keywords with " "; press Enter to separate filter tags										
<input type="checkbox"/>	Subscription ID / Name	Monitor...	Configurations	Database	Consumpt...	SDK Consumption ...	Billing Mo...	Creation ...	Expiration...	Operation
<input type="checkbox"/>	subs-7 j0	Task running	Topic: topic-subs-7as cdb-d... Source Instance ID: cdb-...	MySQL	--	--	Pay as you go	2021-07-28 16:16:19	--	View Subscription Details More

2. Switch between different tabs to view the specific information of subscription details, subscription objects, consumption management, and monitoring data.

**Subscription details**

Subscription Object

Consumption Management

Monitoring Data

Task Log

**Basic Info**

Subscription ID / Name subs-7a0 (k al)

Subscription Type Full

Source Instance Type MySQL

Source Instance ID cdb-f

Tag --

**Subscription Information**

Subscription Topic topic-subs-7a0

Private IP guangzhou-tcs.com:32129

Number of Kafka Partitions 1

Default Kafka Partitioning Strategy By Table Name

**Billing Details**

Region of Source Instance South China (Guangzhou)

Creation Time 2021-07-28 16:16:19

Billing Mode Pay as you go

# Modifying Subscribed Object

Last updated : 2021-02-22 10:05:17

Data subscription Kafka edition supports dynamic increase or decrease of the subscribed objects during data consumption. This document describes how to modify the subscribed object of the data subscription Kafka edition in the console.

## Prerequisites

- You have created the [Data Subscription Kafka Edition](#).

## Directions

- Log in to the [DTS console](#), select **Data Subscription** on the left sidebar to go to the **Data Subscription** page.
- In the data subscription list, select a data subscription and select **More > Modify subscribed object** in the **Operation** column to go to the **Configure data subscription** page.
- In the **Configure data subscription** page, select the subscription type, edit the subscribed object, and click **Save**.
- Return to the subscription list, the subscription instance enters the “enabling” state, and the task is pre-checked and initialized. After being enabled, the subscription instance enters the running state, and the Kafka client can be used to consume the subscription data.



# Pay-as-You-Go to Monthly Subscription

Last updated : 2024-07-08 19:34:21

## Overview

DTS allows you to change pay-as-you-go tasks to monthly subscribed ones. After you perform the billing mode change operation, DTS will generate a monthly subscription renewal order, and you need to pay for the order for the change to take effect.

## Directions

1. Log in to the [DTS console](#) and select **Data Subscription** on the left sidebar to enter the data subscription page.
2. Select the target instance in the instance list, and click **More > Postpaid-to-Prepaid** in the **Operation** column.

<input type="checkbox"/> 订阅 ID / 名称	监控 / 状态 ▾	订阅配置	数据库 ▾	消费时间起点	SDK 当前消费时间位点	计费模式 ▾	创建时间 ↓	到期时
<input type="checkbox"/> su- jos- Kafka 版	 任务运行	Topic: topic- c- 源实例: c- (运行 中)	MySQL	--	--	按量计费	2023-06-13 16:35:34	--
<input type="checkbox"/> su- roc- Kafka 版	 任务运行	Topic: top- f4554u19 源实例: f- 3	MySQL	--	--	按量计费	2023-06-06 14:53:02	--

3. In the pop-up window, select the renewal period, confirm that everything is correct, and click **OK**.

订阅任务按量计费转包年包月

×

您已选1个任务，[收起详情](#) ▼

任务 ID	任务名	计费模式
sut	j	按量计费

续费时长

按年

▼

1

▼

自动续费

☐ 账户余额足够时，设备到期后按月自动续费

原价

¥

优惠价

¥

☐ 已阅读并同意

[按量计费转包年包月规则](#)

🔗

立即转换

取消

# Resetting Subscription

Last updated : 2024-07-08 17:26:21

Data subscription Kafka edition supports resetting the subscription task to remove the information and data of the current subscription instance, and reconfiguring the data subscription task. This document describes how to reset the data subscription in the console.

## Prerequisites

You have created the [Data Subscription Kafka Edition](#).

The state of the data subscription is “running” or “abnormal”.

## Directions

1. Log in to the [DTS console](#), select **Data Subscription** on the left sidebar to go to the **Data Subscription** page.
2. In the data subscription list, select a data subscription and select **More > Reset subscription** in the **Operation** column.
3. In the pop-up dialog box, confirm that everything is correct and click **Confirm**.

### Note:

After the subscription is reset, the binding relationship between the subscription instance and the source instance will be unassociated, and the state of the instance will become “not started”. You can perform the initialization configuration again.

The subscription to the incremental data of the source database will be stopped once the subscription is reset, and the incremental data stored in the subscription will be deleted.

4. In the data subscription list, click **Configure Subscription** in the **Operation** column to reconfigure the subscription task.

# Terminating/Returning an Instance

Last updated : 2024-07-08 17:27:44

## Overviews

**Termination:** For tasks that have ended or failed and are no longer needed, users can destroy them. Once destroyed, these tasks will no longer exist or generate billing.

**Return:** For tasks purchased with an annual or monthly plan that users no longer need, they can be returned.

After a task is terminated or returned, it will enter the Recycle Bin and cannot be recovered. The Recycle Bin retains the task for 7 days before it automatically goes offline, so please proceed with caution.

## Directions

1. Log in to [DTS Console](#), choose **Data Subscription** page in the left sidebar, choose the specific task, and select **More > Terminate/Return** in the **Operation** column.
2. In the pop-up dialog box, after confirming that everything is correct, check the box for **I have read and agree to Termination Rules**, and then click **Terminate Now**.

# Consumption Management

## Creating Consumer Group

Last updated : 2023-10-20 16:35:30

Data subscription Kafka Edition allows users to create multiple consumer groups for multi-point consumption. By creating multiple consumption groups, you can consume for multiple times and increase consumption channels to reduce usage costs and improve the data consumption speed.

## Prerequisites

- You have created the [Data Subscription Kafka Edition](#).

Note :

In the data subscription list, the subscription with the "Kafka Edition" tag is the data subscription in Kafka edition.

## Notes

- The data subscription task must be in the running status.
- A data subscription task can contain up to 10 consumer groups.
- A consumer group can only have one consumer for consumption.

## Directions

- Log in to the [DTS console](#), select **Data Subscription** on the left sidebar to go to the **Data Subscription** page.
- In the data subscription list, select a data subscription and click its name or **View Subscription Details** in the **Operation** column to go to the subscription management page.
- In the subscription management page, select the **Consumption Management** tab and click **Create Consumer Group**.
- In the pop-up window, set the consumer group information and click **Create**.

- Consumer Group Name: set the consumer group name as needed.

- Account: set the consumer group account.
- Password: set the password of the consumer group account.
- Confirm Password: enter the same password again.
- Remarks: set the remarks to record the information.

# Managing Consumer Group

Last updated : 2023-11-16 16:35:20

## Overview

Data Subscription Kafka Edition allows you to manage a consumer group by deleting it and modifying its password or consumption offset.

In addition, you can view information such as client ID (the ID of the client consuming the data), partition ID, consumption offset, consumption offset write time, and consumption delay in **Consumption Management**.

## Prerequisite

You have created a consumer group as instructed in [Creating Consumer Group](#).

## Note

If other Kafka clients are consuming the data in the same consumer group, the consumption offset cannot be modified.

## Modifying Consumer Group Password

1. Log in to the [DTS console](#) and select **Data Subscription** on the left sidebar to enter the data subscription page.
2. In the data subscription list, select the target data subscription, click the subscription name or **View Subscription Details** in the **Operation** column.
3. On the subscription management page, select the **Consumption Management** tab and click **Change Password** in the **Operation** column.
4. In the pop-up window, change the password and click **Modify**.

## Deleting a Consumer Group

1. Log in to the [DTS console](#) and select **Data Subscription** on the left sidebar to enter the data subscription page.
2. In the data subscription list, select the target data subscription, click the subscription name or **View Subscription Details** in the **Operation** column.

3. On the subscription management page, select the **Consumption Management** tab and click **Delete** in the **Operation** column.
4. In the pop-up window, confirm that everything is correct and click **OK**.

**Note**

After the consumer group is deleted, the consumption offset in the consumer group will be deleted, but the data in the data subscription will not. Proceed with caution.

## Modifying the Consumption Offset

1. Log in to the [DTS console](#) and select **Data Subscription** on the left sidebar to enter the data subscription page.
2. In the data subscription list, select the target data subscription, click the subscription name or **View Subscription Details** in the **Operation** column.
3. On the subscription management page, select the **Consumption Management** tab, select the target consumer group, and click **Modify Offset**.
4. In the pop-up window, select the partition for which you want to modify the offset and click **Next**.
5. Set the reset method of data consumption and click **Complete**.

From latest offset: Reset the consumption offset to the latest offset of Kafka messages.

From start offset: Reset the consumption offset to the earliest offset from where Kafka messages start to be stored.

From specified time point: You can enter a time, and DTS will find the consumption offset corresponding to the entered time to start consumption.

If you configure a time out of the offset range, the messages will be consumed from the start or latest offset, whichever is closest to the time you entered.



# Modify Consumption Site offset

Last updated : 2024-07-08 17:29:36

## Overview

This operation guides users to modify the consumption offset.

## Directions

1. Log in to the [DTS console](#) and select **Data Subscription** on the left sidebar to enter the data subscription page.
2. In the data subscription list, select the target data subscription, click the subscription name or **View Subscription Details** in the **Operation** column.
3. On the subscription management page, select the **Consumption Management** tab, select the target consumer group, and click **Modify Offset**.
4. In the pop-up window, select the partition for which you want to modify the offset and click **Next**.
5. Set the reset method of data consumption and click **Complete**.

From latest offset: Reset the consumption offset to the latest offset of Kafka messages.

From start offset: Reset the consumption offset to the earliest offset from where Kafka messages start to be stored.

From specified time point: You can enter a time, and DTS will find the consumption offset corresponding to the entered time to start consumption.

If you configure a time out of the offset range, the messages will be consumed from the start or latest offset, whichever is closest to the time you entered.