TencentDB for TcaplusDB

Quick Start

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Quick Start
- Basic Concepts
  - Cluster
  - Table group
  - Table
  - Index
  - Data Type
  - Read/Write Capacity Mode
  - Table Definition in ProtoBuf
  - TDR table definition
- Creating a Cluster
- Creating Table Group
- Creating Table
- Get connection information
- Accessing TcaplusDB
- External access to TcaplusDB

Quick Start
Basic Concepts
Cluster

Last updated: 2024-10-15 17:11:43

Cluster Overview

The cluster is the basic management unit of TcaplusDB, responsible for providing independent TcaplusDB services for business. From the perspective of game business, a cluster can serve an independent submodule of a game or the entire game.

Cluster Creation

For creation operations, see Create Cluster.

Cluster Details

Users can view the cluster's configuration and attributes through the cluster details to get an overall understanding of the cluster's usage status.
You can log in to TcaplusDB Console, click the Cluster ID, and enter the cluster details page to view the detailed information. There are three main aspects involved:

Basic Information

- Cluster ID: The unique ID of the cluster. This ID cannot be used for database connections but can be used to locate the cluster and control access to it.
- Access ID: The connection ID of the cluster. This ID is used to identify the cluster when accessing the database through SDK and client tools.
- Connection Protocol: Identifies the table description protocol supported by the cluster. Currently supports Protocol Buffers and TDR protocol.
- Cluster Name: The name of the cluster. Users can modify this name based on the cluster usage situation.
- Region: The region where the cluster is located.
- RESTful API: The address used to access the database through the RESTful interface in the same VPC subnet scenario.

Network information

- Network: The VPC network information where the current cluster is located.
- Subnet: The subnet where the current cluster is located.
- Private network address: The internal IP address assigned to the current cluster. CVMs in the same VPC subnet can access this IP address.
- Private network port: The access port number assigned to the current cluster.

Other Information

- Creation Time: The creation time of the current cluster.
- Connection password: If the user forgets the password, it can be viewed through the console to check the cluster's access password.

Table group

Last updated: 2024-10-15 17:11:30

Table Group Overview

As a logic isolation method in TcaplusDB, a table group represents a data partition that can separate different tables from each other. A cluster can contain multiple table groups, and a table group can contain multiple tables. Different table groups cannot access data of each other.
From the perspective of gaming business, if a gaming business supports unified servers in all zones and can read/write the same table group in the cluster, then it does not need to be divided into different zones. It can also be server-specific/zone-specific, i.e., the cluster can contain multiple table groups.

Creating Table Group

For create operations, see Create Table Group.

Table Group Details

Users can view the configuration and properties of table groups through the cluster list page for a comprehensive understanding of the cluster's usage.
You can log in to TcaplusDB Console, enter the cluster list, and thus view the table group information under the specified cluster.
Table group information contains the following four fields:

- ID: it is the ID of the table group in the current cluster, which is required during database connection. Please note that the table group IDs may be repeated in different clusters.
- Table Group Name: you can customize the name of the table group based on its actual purpose.
- Table Count: it indicates the number of tables in the current table group.
- Total Capacity: it indicates the disk capacity used by the tables in the current table group.

Table

Last updated: 2024-10-15 17:12:11

Table Overview

A Table is a collection of data, which serves as a combined form where business data is stored together according to the Table's Definition.
From a game business perspective, a Table can be responsible for a set of related data for a business module, such as a User Table or Item Table.
TcaplusDB requires a preset Schema and provides two types of tables: Generic Table and List Table. A Generic Table uses one Key (1-8 fields) to correspond to one Value Record, while a List Table uses one Key (1-7 fields) to correspond to multiple Value Records.
List Tables are divided into Ordinary List Table and SortList Table based on whether they are sorted:

- For an Ordinary List Table, Tcaplus inserts data in an unordered manner. When a user retrieves all Values under a Key, the Values are unordered.
- A SortList Table sorts the data based on a specified Value field, allowing users to retrieve values in ascending or descending order. PB and TDR tables both support the Generic Table, List Table, and SortList Table schemas. Users can choose the appropriate schema according to their needs.

In the following example of Player Table data, the table can be used to store information about all players in a game, such as Game Character Name, Gender, Race, Level, Combat Value, Equipment, Mount, and Items.

```
{
player_id:11474,
player_name: "Test Account 2",
gender:0,
ethnicity:"Elf",
FightingPower:10,
equipment:
    {
        helmet:0,
        Warframe:0,
        gloves:0,
        necklace:0,
        pants:0,
        Shoes:0
    },
horse:"0"
},
{
player_id:11475,
player_name: "Test Account 1",
gender:1,
ethnicity:"Orc",
FightingPower:1477,
equipment:
    {
        helmet:1478,
        Warframe:21,
        gloves:554,
        necklace:12,
        pants:64,
        Shoes:122
    },
horse:"3"
}
```

Record

A collection of multiple field values of the same object is called a Record. In the above example of the Player Character Table, all information of a player character, composed of multiple attribute field values, represents the character's information in the game. A single record supports a maximum of 10MB.

Field

A field describes a certain characteristic of a specified object. In the above example, an individual attribute of the player character is referred to as a field, such as player_name and ethnicity in the player character table.

Primary Key Field

As can be seen from the above example table, each record has a field that marks the uniqueness of the record, used to distinguish records in the table. We call it the primary key field. In the player character table of the above example, the primary key fields include player_id and player_name. TcaplusDB supports up to 8 primary key fields.

Common Field

In addition to the fields marking the uniqueness of each record, there are other attributes. The fields constituted by these attributes are called common fields. In the player character table of the above example, the common fields are attributes like equipment and FightingPower. TcaplusDB supports up to 128 common fields.

Shard Field

TcaplusDB is a distributed database system that can provide higher concurrency request capability for tables. If the user sets shard fields, TcaplusDB will automatically adjust the table shards based on the current table access situation by hashing the shard fields. If the user does not set them, the system will default to using the primary key fields as shard fields.
The setting of shard fields is related to the distribution of background data. Users need to evaluate whether the values of the fields used as shard fields are discrete. Fields with limited values such as gender or weekdays are not recommended as shard fields. It is recommended to use fields like ID and name as shard fields.

Field Type

TcaplusDB supports multiple data types. The supported types are as follows:

Field Type	Field Introduction
int32	Using variable length encoding, this type represents a signed integer between -2^31 and +2^31, but the efficiency of representing negative numbers is low.
int64	Using variable length encoding, this type represents a signed integer between -2^63 and +2^63.
uint32	Using variable length encoding, this type represents an unsigned integer value from 0 to 2^32.
uint64	Using variable length encoding, this type represents an unsigned integer value from 0 to 2^64.
sint32	Variable length signed int, more efficient at representing negative values compared to regular int32.
sint64	Variable length signed int, more efficient at representing negative values compared to regular int64.
bool	Boolean type, represents True or False.
fixed32	Fixed-length numeric type, always occupies four bytes. More efficient than uint32 if values are generally greater than 2^28.
fixed64	Fixed-length numeric type, always occupies eight bytes. More efficient than uint64 if values are generally greater than 2^56.
sfixed32	Fixed-length numeric type, always occupies four bytes.
sfixed64	Fixed-length numeric type, always occupies eight bytes.
float	Floating point type, occupies 4 bytes, represented by 32-bit binary.
double	Double precision type, occupies 8 bytes, represented by 64-bit binary.
string	String type, only UTF-8 encoded or 7-bit ASCII text is allowed, and the length cannot exceed 2^32.

bytes	Binary type, can contain a byte sequence of any length no longer than 2^32.
message	Nested type, can be continuously nested, supports up to 128 levels of nesting. Excessive nesting affects performance.

Table Definition format

TcaplusDB supports Protocol Buffers, TDR data storage format. It is a lightweight and efficient structured data storage format mainly used for serializing structured data.
For more details about the table format, you can check Table Description Language.

Generic Table

Basic Characteristics

- Indexes can be created. Please refer to Index.
- Supports up to 8 Key fields and 256 Value fields.
- The maximum size of a single Key is 1KB, the maximum size of a single Value is 10MB, and the maximum size of an entire record is 10MB.

TDR-Generic Table Example

```
<struct name="PLAYERONLINECNT" version="1" primarykey="TimeStamp,GameSvrID" splittablekey="TimeStamp">
    <entry name="TimeStamp"           type="uint32"     desc="Unit is minutes" />
    <entry name="GameSvrID"         type="string"     size="64" />
    <entry name="GameAppID"         type="string"     size="64" desc="gameapp id" />
    <entry name="OnlineCntIOS"      type="uint32"     defaultvalue="0" desc="iOS Online User Count" />
    <entry name="OnlineCntAndroid"  type="uint32"     defaultvalue="0" desc="Android Online User Count" />
    <entry name="BinaryLen"         type="smalluint" defaultvalue="1" desc="Data source length; if length
is 0, source check is ignored"/>
    <entry name="binary"            type="tinyint"     desc="Binary" count= "1000" refer="BinaryLen" />
    <entry name="binary2"           type="tinyint"     desc="Binary2" count= "1000" refer="BinaryLen" />
    <entry name="strstr"            type="string"     size="64" desc="String"/>
    <index name="index_id"  column="TimeStamp"/>
</struct>
```

PB-Generic Table Example

```
message pb_generic_index_shardingkey {
    option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey,svrid";
    option(tcaplusservice.tcaplus_index) = "index_openid(openid)";
    option(tcaplusservice.tcaplus_index) = "index_openid_tconndid(openid,tconndid)";
    option(tcaplusservice.tcaplus_index) = "index_full_key(openid,tconndid,timekey,svrid)";

    option(tcaplusservice.tcaplus_sharding_key) = "openid";

    // 4 keys
    required uint32 openid = 1; //QQ Uin
    required string timekey = 2[(tcaplusservice.tcaplus_size) = 20, (tcaplusservice.tcaplus_desc) =
"bingo"];
    required int32 tconndid = 3;
    required string svrid = 4;

    //value
    required string gamesvrid = 5[(tcaplusservice.tcaplus_size) = 11];
    repeated property other_property= 6 ;//Other extended properties
    optional item_bag items = 7;
    repeated int64 lockid = 8 [packed = true];
    optional bytes val = 9;
    optional pay_info pay = 10;
```

```
    optional uint32 id_uint32 = 11;
    optional int32 id_int32 = 12;
}
```

List Table

Basic Characteristics

- Indexes cannot be created.
- Key fields can have up to 7, and Value fields can have up to 255, with one reserved for system use.
- List supports convenient head and tail operations, suitable for scenarios like Email, Message Board, Battle Record, etc.
- The List Table needs to specify the maximum number of elements under a single Key (currently a maximum of 10,000). When the limit is exceeded, old elements can be deleted from the header or tail.
- All Values under a single Key can be retrieved at once, with the internal arrangement of Values being unordered.
- Tcaplus uses a grading mechanism for storing the List Table, including:
  - Index Record, Fullkey + (idx1, idx2, idx3, ···, idxn)
  - Data Record, [FullKey + idx1, value1][FullKey + idx2, value2][···][FullKey + idxn, valuen]

TDR-List Table Example

```
<struct name="following_action_list" version="1" primarykey="game,myuin">
    <entry name="game" type="uint64" defaultvalue="0" desc="Game ID"/>
    <entry name="myuin" type="uint32" desc="QQ Number"/>
    <entry name="actiontype" type="uint8" defaultvalue="0" desc="1-Share Image, 2-Like Image, 3-Comment on
Image"/>
    <entry name="uin2" type="uint32" desc="Follow Person's QQ Number"/>
    <entry name="nick2" type="string" size="128" desc="Follow Person's Nickname"/>
    <entry name="sex" type="uint8" defaultvalue="0" desc="Follow Person's Gender Male 0 Female 1"/>
    <entry name="pid" type="string" size="33" desc="Follow Person's Operated Image ID"/>
    <entry name="time" type="uint32" desc="Time"/>
    <entry name="content" type="string" size="1024" desc="Dynamic Detailed Content"/>
</struct>
```

PB-List Table Example

```
syntax = "proto3";

package myTcaplusTable;

import "tcaplusservice.optionv1.proto";

message tb_online_list {
    option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey";
    option(tcaplusservice.tcaplus_customattr) = "TableType=LIST;ListNum=1900";

    int32 openid = 1; //QQ Uin
    int32 tconndid = 2;
    string timekey = 3;
    string gamesvrid = 4;
    int32 logintime = 5 ;
    repeated int64 lockid = 6;
    pay_info pay = 7;

    message pay_info {
        uint64 total_money = 1;
        uint64 pay_times = 2;
    }
    map<string, pay_info> projects = 8;
```

```
}
```

SortList Table

Basic Characteristics

Up to 4 sorting fields allowed, specified in the XML (tdr) when creating the table. The format is as follows:

```
<struct name="following_action_list" version="1" primarykey="game,myuin"
customattr="TableType=SORTLIST;SortRule=INSC;SortFieldNum=1">
    <entry name="game" type="uint64" defaultvalue="0" desc="Game ID"/>
    <entry name="myuin" type="uint32" desc="QQ Number"/>
    <entry name="actiontype" type="uint8" defaultvalue="0" desc="1-Share Image, 2-Like Image, 3-Comment on
Image"/>
    <entry name="uin2" type="uint32" desc="Follow Person's QQ Number"/>
    <entry name="nick2" type="string" size="128" desc="Follow Person's Nickname"/>
    <entry name="sex" type="uint8" defaultvalue="0" desc="Follow Person's Gender Male 0 Female 1"/>
    <entry name="pid" type="string" size="33" desc="Follow Person's Operated Image ID"/>
    <entry name="time" type="uint32" customattr="sort1" desc="Time"/>
    <entry name="content" type="string" size="1024" desc="Dynamic Detailed Content"/>
</struct>
```

> **Note:**
> As above, customattr="TableType=SORTLIST;ListNum=1023;SortFieldNum=1" indicates the table type is SORTLIST, SortRule=INSC indicates ascending order, SortFieldNum=1 indicates there is 1 sorting field, and customattr="sort1" indicates the first sorting field.

Use Limits

- The default sorting is in ascending order.
- It is not allowed to change from an unordered List to a SortList or from a SortList to an unordered List during table alterations. Changing the order of sorting fields or adding/removing sorting fields is also not allowed (implemented via custom Key-Value table alterations using `so`).
- The maximum byte length for sorting fields is 8B. Sorting field types: byte, uint16, uint32, uint64, int16, int32, int64, float, double [string (including \0 up to 8B, not supported at the moment)].
- Sorting refers to the Value fields participating in the sorting, not the Key.

Usage Instructions

- Sorting: After an existing data structure is sorted, using ListAddAfter for data insertion yields the best results with insertion sort.
- Insert
- In the case of ListAddAfter, the process is: first check if it is full. If it is full and element elimination is not allowed, the insertion fails. If it is full and elimination is allowed, delete an element, obtain a BiggestIndex, insert the new element at the corresponding position, and shift the other elements.
- For SortList, when the user performs a ListAddAfter, the interface description in ServiceApi needs to be updated since the default sorting is in ascending order. To eliminate the last element means eliminating the largest element, and to eliminate the first element means eliminating the smallest element.

TDR-Sortlist Table Example

```
<struct name="table_Sortlist_single" primarykey="key, name"
customattr2="TableType=SORTLIST;ListNum=1023;SortFieldNum=1;SortRule=DESC" version="5" desc="Used for list
table traversal testing, requires 4 shards, with a list maximum of 1023 elements" >
        <entry name="key" type="uint32" desc="When using a single uint32 as the KEY, hashcode = key %
10000"/>
        <entry name="name"     type="int16" />
        <entry name="level"      type="uint32" />
        <entry name="value1" type="string" size="102400" defaultvalue="" desc="Maximum length: 100KB"/>
```

```
        <entry name="value2" type="string" size="102400" defaultvalue="" desc="Maximum length: 100KB"/>
        <entry name="type_int8" type="int8" desc="type_int8" />
        <entry name="type_uint8" type="uint8" desc="type_uint8"/>
        <entry name="type_int16" type="int16" desc="type_int16" customattr2="sort1"/>
        <entry name="type_uint16" type="uint16" desc="type_uint16"/>
        <entry name="type_int32" type="int32" desc="type_int32"/>
        <entry name="type_uint32" type="uint32" desc="type_uint32"/>
        <entry name="type_int64" type="int64" desc="type_int64"/>
        <entry name="type_uint64" type="uint64" desc="type_uint64"/>
        <entry name="type_float" type="float" desc="type_float"/>
        <entry name="type_double" type="double" desc="type_double"/>
        <entry name="type_short" type="short" desc="type_short"/>
        <entry name="type_string" type="string" desc="type_string" size="20"/>
        <entry name="type_tinyint" type="tinyint" desc="type_tinyint"/>
        <entry name="type_datetime" type="datetime" desc="type_datetime"/>
</struct>
```

PB-SortList Table Example

```
message pb_sortedlist {
    option(tcaplusservice.tcaplus_primary_key) = "openid,tconndid,timekey,svrid";
    option(tcaplusservice.tcaplus_customattr) = "TableType=SORTLIST;ListNum=1900;SortField=id_int32";

    // 4 keys
    required uint32 openid = 1; //QQ Uin
    required string timekey = 2[(tcaplusservice.tcaplus_size) = 20, (tcaplusservice.tcaplus_desc) =
"bingo"];
    required int32 tconndid = 3;
    required string svrid = 4;

    //value
    required string gamesvrid = 5[(tcaplusservice.tcaplus_size) = 11];
    repeated property other_property= 6 ;//Other extended properties
    optional item_bag items = 7;
    repeated int64 lockid = 8 [packed = true];
    optional bytes val = 9;
    optional pay_info pay = 10;
    optional uint32 id_uint32 = 11;
    optional int32 id_int32 = 12;
}
```

Index

Last updated: 2024-10-15 17:12:34

Index Overview

You can create an index for a table and use it to quickly get desired data from a database, which delivers higher flexibility in data query for your application.
TcaplusDB supports two types of indexes:

- Local index: it must be composed of primary key fields and allows you to quickly query data by using a primary key as a filter.
- Global Index: Supports fields composed of any types except message type, and allows for fast query using the configured fields in the global index as conditions.

TcaplusDB supports up to 8 local indexes.

Local index

Features

- Local Index is a real-time index, which updates index data simultaneously when inserting or deleting data.
- The fields of a Local Index must be included in the Primary Key Fields and must include the Sharding Field; thus, the query will ultimately be performed on a single data shard.
- Local Index only supports Equi-join Query.
- A table can have multiple Local Indexes; the query must include all fields of one of the Local Indexes.
- Currently, only Generic Table supports Local Index.

Local Index can only be defined during table creation and must be composed of Primary Key Fields. The intersection of all Index Field Collections cannot be empty.

Example

For example, a table may have 4 Primary Key Fields, as shown in the following HeroInfo table: heroId, heroName, heroFightingType, and heroQuality are the Primary Key Fields of this table.

```
{
    heroId:1,
    heroName:"Arthur",
    heroFightingType:1,
    heroQuality:3
    heroSkill:{
        BasicSkill1:1,
        BasicSkill2:2,
        SpecialSkill:3
    },
    heroLevel:12,
    heroskin:2,
    heroAttackpower:141,
    heroPhysicalDefense: 283,
    heroMagicdefense:124
}
{
    heroId:4,
    heroName:"Shooter",
    heroFightingType:3,
    heroQuality:4
    heroSkill:{
        BasicSkill1:1,
        BasicSkill2:2,
        SpecialSkill:3
    },
    heroLevel:11,
```

```
    heroskin:1,
    heroAttackpower:225,
    heroPhysicalDefense: 57,
    heroMagicdefense:41
}
```

For the above table, you can freely combine the 4 Primary Key Fields to create indexes, but the created indexes' intersection must not be empty. For example, using heroId as the intersection field, you have the following combination options for index design:

- heroId,heroName
- heroId,heroFightingType
- heroId,heroQuality
- heroId,heroName,heroFightingType
- heroId,heroFightingType,heroQuality
- heroId,heroName,heroFightingType,heroQuality

Additionally, you can query data based on the above-created index fields. When creating indexes, please do so according to the actual requirements of your business; otherwise, performance will be reduced due to Index Redundancy.

Global Index

In the previously displayed HeroInfo table, you can query data items based on the fields contained in the already created Local Indexes. If you also want to query data based on information such as heroLevel and heroSkin, you can add heroLevel and heroSkin fields to the Global Index, then query the data using the fields contained in the Global Index.
Please note that Global Indexes also do not support Nested Types, such as the heroSkill field in the example table above.

Use Limits

Local Index Use Limits

- Once created, Local Indexes cannot be modified, deleted, added, or deleted during use; they are deleted with the table.
- The number of Local Indexes cannot exceed 8.
- Local Indexes only support Exact Match. That is, when using local index fields as query conditions, you can only exactly match specific values; Fuzzy and Range Matching are not supported.

Global Secondary Index Use Limits

- Global Secondary Indexes only support Table-level Fields. Nested Fields and Array List Type Fields do not support the creation of secondary indexes.
- Global Secondary Indexes only support Generic Type Tables and do not support List Type Tables.
- Global Secondary Indexes can only be used in the tcaplus_client Tool, C++ SDK (TDR Protocol Table & PB Protocol Table), and are currently not supported at the API Level for non-C++ languages.

Data Type

Last updated: 2024-10-15 17:13:04

TencentDB for TcaplusDB supports Protocol Buffers data types.

Mapping between proto3 and Data Types in Programing Languages

.proto type	C++ type	Java type	Python type	Go type	Ruby type	C#	PHP type	Dart	Description
double	double	double	float	float64	float	double	float	double	-
float	float	float	float	float32	float	float	float	double	-
int32	int32	int	int	int32	fixnum/bignum (as needed)	int	integer	int	Uses variable-length encoding. Inefficient encoding of negative numbers. If your field might have negative values, use sint32 instead.
int64	int64	long	int/long	int64	bignum	long	integer/string	Int64	Uses variable-length encoding. Inefficient encoding of negative numbers. If your field might have negative values, use sint64 instead.
uint32	uint32	int	int/long	uint32	fixnum/bignum (as needed)	uint	integer	int	Uses variable-length encoding.
uint64	uint64	long	int/long	uint64	bignum	ulong	integer/string	Int64	Uses variable-length encoding.
sint32	int32	int	int	int32	fixnum/bignum (as needed)	int	integer	int	Uses variable-length encoding.
sint64	int64	long	int/long	int64	bignum	long	integer/string	Int64	Uses variable-length encoding. Signed int values. They encode negative numbers more efficiently than regular int32.
fixed32	uint32	int	int/long	uint32	fixnum/bignum (as needed)	uint	integer	int	Always four bytes. More efficient than uint32 if values are generally greater than 2^28.
fixed64	uint64	long	int/long	uint64	bignum	ulong	integer/string	Int64	Always eight bytes. More efficient than uint64 if values are generally greater than 2^56.
sfixed32	int32	int	int	int32	fixnum/bignum (as needed)	int	integer	int	Always four bytes.
sfixed64	int64	long	int/long	int64	bignum	long	integer/string	Int64	Always eight bytes.
bool	bool	boolean	bool	bool	trueClass/falseClass	bool	boolean	bool	-

string	string	String	str/unicode	string	string(UTF-8)	string	string	string	A string must always contain UTF-8 encoded or 7-bit ASCII text, and cannot be longer than 2^32.
bytes	string	ByteString	str	byte	string(ASCII-8BIT)	bytestring	string	–	May contain any arbitrary sequence of bytes no longer than 2^32.

Mapping between proto2 and Data Types in Programing Languages

.proto type	C++ type	Java type	Python type	Go type	Description
double	double	double	float	*float64	–
float	float	float	float	*float32	–
int32	int32	int	int	*int32	Uses variable-length encoding.Inefficient for encoding negative numbers. If your field is likely to have negative values, use sint32 instead.
int64	int64	long	int/long	*int64	Uses variable-length encoding.Inefficient for encoding negative numbers. If your field is likely to have negative values, use sint64 instead.
uint32	uint32	int	int/long	*uint32	Uses variable-length encoding.
uint64	uint64	long	int/long	*uint64	Uses variable-length encoding.
sint32	int32	int	int	*int32	Uses variable-length encoding.Signed int value. These more efficiently encode negative numbers than regular int32s.
sint64	int64	long	int/long	*int64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.
fixed32	uint32	int	int/long	*uint32	Always four bytes.More efficient than uint32 if values are generally greater than 2^28.
fixed64	uint64	long	int/long	*uint64	Always eight bytes.More efficient than uint64 if values are generally greater than 2^56.
sfixed32	int32	int	int	*int32	Always four bytes.
sfixed64	int64	long	int/long	*int64	Always eight bytes.
bool	bool	boolean	bool	*bool	–
string	string	String	unicode (Python 2) or str(Python 3)	*string	A string must always contain UTF-8 encoded or 7-bit ASCII text.
bytes	string	ByteString	bytes	byte	May contain any arbitrary sequence of bytes.

Read/Write Capacity Mode

Last updated: 2024-10-15 17:13:25

Read/Write Capacity Mode Overview

TcaplusDB uses the read/write capacity mode to calculate the volume of table read/write requests.

Capacity unit (CU)

It is the minimum request unit of resource access and the minimum statistical unit of requested data reads/writes.

Read capacity unit (RCU)

It is the minimum unit of read requests. The response data generated for a request may contain multiple RCUs. The maximum value of an RCU is 4 KB, i.e., an RCU indicates that a read request can be performed on a record that is up to 4 KB in size. If you need to read data of more than 4 KB, you need more RCUs.
The base number of an RCU is 4 KB. The size of a request below 4 KB is calculated as 4 KB, and the size of a request above 4 KB is calculated as a multiple of 4 KB (the remaining part below 4 KB is calculated as 4 KB).
Below is an example:
Read request capacity
A user sends a request to query a field in a record which is 10 KB in size; the database returns the value of the specified field of the record. In this case, the response data is 10 KB, which is 3 RCUs.

Write Capacity Unit (WCU)

It is the smallest unit of write requests. The written data generated for a write request may contain multiple WCUs. The maximum value of a WCU is 4 KB, i.e., every 4 KB of written data will be counted as one WCU.
The base number of a WCU is 4 KB. The size of a request below 4 KB is calculated as 4 KB, and the size of a request above 4 KB is calculated as a multiple of 4 KB (the remaining part below 4 KB is calculated as 4 KB).
When performing a write operation, you can select whether to enable the write operation return feature (Result Flag). If it is enabled, the system will automatically calculate the returned CUs as WCUs.
Consumed number of WCUs = CUs generated by data write + returned CUs
For example:

- **Write request**
  A user sends a request to update content of 5 KB, and the backend reads the record (50 KB in total) containing the target data from the disk into the memory.
  Then, the backend updates the record and flushes the 50 KB content into the disk after the update.
  If the user enables the write operation return record, 5 KB of content will be returned to the user.
- **Result flag enabled**
  13 CUs (50 KB data written into disk) + 2 CUs (5 KB returned data) = 15 WCUs
- **Result flag not enabled**
  13 CUs (50 KB data written into disk) = 13 WCUs

Preset mode

If you select the preset mode, you need to adjust the preset read/write capacity of the table based on access change. This operation can help you control the TcaplusDB use to keep it at or below the defined request rate, so that you can predict the costs. To better configure the preset capacity, you need to estimate the application traffic in advance.

Reserved read

It refers to the RCU preconfiguration of a table. You need to estimate the highest RCU consumption per second of the table on the day and set the value as the reserved RCUs.

Reserved write

It refers to the WCU preconfiguration of a table. You need to estimate the highest WCU consumption per second of the table on the day and set the value as the reserved WCUs.

Reserved capacity

It refers to the preconfiguration of disk capacity usage of a table. You need to estimate the capacity used by the table on the day.

Request limit and quota

As your configured RCU or WCU value may be too low or the number of access requests may surge, TcaplusDB limits the requests exceeding 40% of the reserved RCUs or WCUs by default. This limit does not block the requests; instead, it makes the requests queue up to reduce the access frequency, which may cause timeout in the program. You can use the monitoring and alarming feature to know whether the current number of requests exceeds the preconfigured capacity.
This limit can prevent your application from consuming too many CUs. However, if requests are limited, they will fail, and many system errors will occur.
You can use TcaplusDB table monitoring to monitor the actual RCUs/WCUs and modify the table quota if necessary.

Table Definition in ProtoBuf

Last updated: 2024-10-15 17:13:56

TcaplusDB supports two table definition formats: Protobuf (Protocol Buffers, PB) and Tencent Data Representation (TDR).
Both formats are equally effective. Choose the table definition language based on your specific business usage habits.

- Protobuf is a descriptive language developed by Google for serializing structured data, emphasizing simplicity and performance.
- TDR is a cross-platform data representation language developed by Tencent, combining the advantages of XML, binary, and ORM (Object-Relational Mapping), and is widely used in the serialization of game data by Tencent.

This document primarily describes the table definition format of Protobuf.

Protobuf Table

To create a table in TcaplusDB, you first need to use a table description language to define the table format. Write the table definition content into a file, which is called the table's IDL description file.
The IDL description file of a Protobuf table is defined according to the rules of Protocol Buffers.
The main categories of definition are:

- File Definition Information
- Table Definition
- Nested Type Information

File Definition Information

The File Definition Information section principally defines the public information for the current table description file, primarily involving three types of content:

Option Name	Feature Description	Sample Value	Required
syntax	Indicates the syntax specification version of the current file.	Supports proto2, proto3	Yes
package	Specify the current file's package name from the Definition. The package name can avoid name conflicts between message types.	Package Name Information	Yes
import	Introduce some public information of the Tcaplus table, which must be referenced in your table Definition.	tcaplusservice.optionv1.proto	Yes

Table Definition

The table Definition information is mainly formatted through message Definition tables. In the message, you can define extended information of the table or define the field information of the table.

Extended Information Definition

TcaplusDB table definitions can be extended through options based on protobuf syntax to achieve richer semantic features. The definable content is shown in the table below.
The detailed definition format is: `option(tcaplusservice.option) = "value";`

Option Name	Feature Description	Configuration Example	Required
tcaplus_primary_key	Set the primary key field for the TcaplusDB table	option(tcaplusservice.tcaplus_primary_key) = "uin,name,region";	Yes
tcaplus_index	Set the index key field for the TcaplusDB table	option(tcaplusservice.tcaplus_index) = "index_1(uin,region)";	No

tcaplus_sharding_key	Users can define shard keys for the table	option(tcaplusservice.tcaplus_sharding_key) = "uin";	No
tcaplus_field_cipher_suite	If you need to use the field encryption feature, please set it using the example; if users need to specify their own encryption algorithm, refer to the example in the API	option(tcaplusservice.tcaplus_field_cipher_suite) = "DefaultAesCipherSuite";	No
tcaplus_cipher_md5	If you need to use the field encryption feature, you need to set the MD5 of the password string saved on the user side	option(tcaplusservice.tcaplus_cipher_md5) = "62fee3b53619b7f303c939964c6f2c4b";	No

Field Information Definition

The format of TcaplusDB definition fields is:

```
Field modifier Field type Field name = Identification Number [special Definition];
```

Field Modifier

Proto2 supports three types of qualifying modifiers, while proto3 no longer supports the 'required' modifier and defaults to 'optional' type.

- required: Indicates that the field is mandatory. In proto2, primary key fields must be marked as required.
- optional: Indicates that the field is optional and supports setting a default value.
- repeated: Indicates that the field can contain 0 – N elements. Its characteristics are similar to optional, but it can contain multiple values at a time, essentially passing an array of values. Must specify the special definition [packed = true].

Field Type

TcaplusDB supports both standard fields and nested fields. For a detailed introduction to fields, refer to Field Types.

Field Name

Name fields according to the current attributes. Supports uppercase and lowercase letters, numbers, and underscores. It is recommended to use camelCase for field naming, and it cannot start with a number.

Identifier

Range of identifier usage: [1, 2^29 – 1], cannot use identifiers [19000–19999] because these have been reserved in the Protobuf protocol implementation. Using these will result in an error.
Each field occupies memory during encoding, and the amount of occupied memory depends on the identifier:

- Fields with identifiers in the range [1, 15] occupy 1 byte during encoding.
- Fields with identifiers in the range [16, 2047] occupy 2 bytes during encoding.

Special Definition

- When a repeated field needs to specify the packed=true option. Usage is as follows:

```
repeated int64 lockid = 6 [packed = true];
```

- Fields marked as optional can have a default value specified using default = 1. Usage is as follows:

```
optional int32 logintime = 5 [default = 1];
```

- Fields of type string and bytes can be designated as encrypted fields. Usage is as follows:

```
required string name = 2[(tcaplusservice.tcaplus_crypto) = true];
```

Nested Type Information

TcaplusDB supports nested types. A nested type can contain another nested type as its field, and a new nested type can also be defined within a nested type.

The definition of a nested type is similar to that of a table; both are declared using message, but the structure cannot include extended information definitions like "primary key" and "index".
TcaplusDB supports up to 128 levels of continuous nesting, but heavy use of nesting is not recommended as too many nesting levels can degrade data access performance.

proto2 Table Description File Example

Here is a table description file that complies with the proto2 syntax specification:

```
syntax = "proto2";                    // Indicates compliance with proto2 syntax specification
package myTcaplusTable;               // Self-defined package name

import "tcaplusservice.optionv1.proto"; // This file defines some common information for Tcaplus tables
and needs to be referenced (imported) in your table definition

message player {  // Use message to define the table, the message name is the table name

    // Only a message with the primary key option (tcaplusservice.tcaplus_primary_key) specified can be
recognized as a TcaplusDB business data table; otherwise, this message is just a structure
    // The primary key option specifies the list of primary key field names, separated by commas, with up
to four fields being specified as primary keys
    option(tcaplusservice.tcaplus_primary_key) = "uin,name,region";

    // The tcaplusservice.tcaplus_index option specifies Tcaplus indexes
    // Each index must include primary key fields, and the intersection of all index field sets must not
be empty
    option(tcaplusservice.tcaplus_index) = "index_1(uin,region)";
    option(tcaplusservice.tcaplus_index) = "index_2(uin,name)";

    // option(tcaplusservice.tcaplus_sharding_key) = "uin"; Users can explicitly set the index sharding
field. If not explicitly set by the user, the system will default to using the primary key fields as
sharding fields

    option(tcaplusservice.tcaplus_field_cipher_suite) = "DefaultAesCipherSuite"; // Use the default
DefaultAesCipherSuite encryption function, optional setting
    option(tcaplusservice.tcaplus_cipher_md5)= "62fee3b53619b7f303c939964c6f2c4b"; // Set the MD5 value
for the encrypted password string, optional setting

    // Next is the table's Field Definition
    // TcaplusDB table supports the following data types:
    // Non-nested Type: int32, int64, uint32, uint64, sint32, sint64, bool, fixed64, sfixed64, double,
fixed32, sfixed32, float, string, bytes
    // Nested Type: message
    // Tcaplus supports REQUIRED, OPTIONAL and REPEATED as Field Modifiers

    // Primary Key Fields (up to 4)
    required int64 uin = 1; // Primary key fields must be marked as required and cannot be of nested type
    required string name = 2 [(tcaplusservice.tcaplus_crypto) = true]; // string and bytes fields in
messages can be designated as encrypted fields, optional setting
    required int32 region = 3;
    // A table can contain up to 4 primary key fields

    // Ordinary Value Field
    required int32 gamesvrid = 4; // Ordinary fields can be marked as required, optional, or repeated
    optional int32 logintime = 5 [default = 1];
    repeated int64 lockid = 6 [packed = true]; // Fields marked as repeated need to have the packed=true
option specified
    optional bool is_available = 7 [default = false]; // Optional fields can specify default values
    optional pay_info pay = 8; // The type of value field can be a self-defined structure type
}
```

```
message pay_info { // Using message definition structure

    required int64 pay_id = 1;
    optional uint64 total_money = 2;
    optional uint64 pay_times = 3;
    optional pay_auth_info auth = 4;

    message pay_auth_info { // Structure type supports nested definition
        required string pay_keys = 1;
        optional int64 update_time = 2;
    }
}
```

proto3 table description file example

The following is an example of a table description file that conforms to the proto3 syntax specification:

```
syntax = "proto3"; // Indicates compliance with the proto3 syntax specification
package myTcaplusTable;                 // Self-defined package name

import "tcaplusservice.optionv1.proto"; // This file defines some common information for Tcaplus tables
and needs to be referenced (imported) in your table definition

message player {  // Use message to define the table, the message name is the table name

    // Only a message with the primary key option (tcaplusservice.tcaplus_primary_key) specified can be
recognized as a TcaplusDB business data table; otherwise, this message is just a structure
    // The primary key option specifies the list of primary key field names, separated by commas, with up
to four fields being specified as primary keys
    option(tcaplusservice.tcaplus_primary_key) = "uin,name,region";

    // The tcaplusservice.tcaplus_index option specifies Tcaplus indexes
    // Each index must include primary key fields, and the intersection of all index field sets must not
be empty
    option(tcaplusservice.tcaplus_index) = "index_1(uin,region)";
    option(tcaplusservice.tcaplus_index) = "index_2(uin,name)";

    // option(tcaplusservice.tcaplus_sharding_key) = "uin"; Users can explicitly set the sharding field
themselves. If the user does not explicitly set it, the system will default to using the primary key field
as the sharding field

    option(tcaplusservice.tcaplus_field_cipher_suite) = "DefaultAesCipherSuite"; // Use the default
DefaultAesCipherSuite encryption function, optional setting
    option(tcaplusservice.tcaplus_cipher_md5)= "62fee3b53619b7f303c939964c6f2c4b"; // Set the MD5 value
for the encrypted password string, optional setting

    // Next is the table's Field Definition
    // TcaplusDB table supports the following data types:
    // Non-nested Type: int32, int64, uint32, uint64, sint32, sint64, bool, fixed64, sfixed64, double,
fixed32, sfixed32, float, string, bytes
    // Nested Type: message

    // Primary Key Fields (up to 4)
    int64 uin = 1; // Primary key fields must be non-nested types
    string name = 2[(tcaplusservice.tcaplus_crypto) = true]; // String and bytes fields in messages can be
designated as encrypted fields, optional setting
    int32 region = 3;
    // A table can contain up to 4 primary key fields

    // Ordinary Value Field
```

```
    int32 gamesvrid = 4;
    int32 logintime = 5;
    int64 lockid = 6;
    bool is_available = 7;
    pay_info pay = 8; // The type of value field can be a self-defined structure type
}

message pay_info { // Using message definition structure

    int64 pay_id = 1;
    uint64 total_money = 2;
    uint64 pay_times = 3;
    pay_auth_info auth = 4;

    message pay_auth_info { // Structure type supports nested definition
        string pay_keys = 1;
        int64 update_time = 2;
    }
}
```

TDR table definition

Last updated: 2024-10-15 17:14:20

TcaplusDB supports two table definition formats: Protobuf (Protocol Buffers, PB) tables and TDR (Tencent Data Representation) tables.

- Protobuf is a descriptive language developed by Google for serializing structured data, emphasizing simplicity and performance.
- TDR is a cross-platform data representation language developed by Tencent, combining the advantages of XML, binary, and ORM (Object-Relational Mapping), and is widely used in the serialization scenarios of Tencent's game data.
  Both formats are fundamentally similar in usage. Choose the table definition language based on your specific business usage habits.

This document mainly describes the definition format of TDR tables.

TDR Table

TDR supports generic tables and list tables. Different types of tables can be created in a single XML file.

- A generic table is a table that represents element attributes in a tabular form, such as students, employers, and game players.
- A list table is a series of records, such as game leaderboards and in-game emails (typically the most recent 100 emails).

Table Definition

- The element metalib is the root element of the XML file.
- The attribute tagsetversion should always be 1.
- A struct element containing a primarykey attribute is a table; a struct element without a primarykey attribute is a regular structure.
- Each time the table structure is changed, the version attribute needs to be incremented by 1. The starting value of version attribute is always 1.
- The primarykey attribute specifies the primary key fields; for generic tables, you can specify up to 4 primary key fields, and for list tables, up to 3.
- The splittablekey attribute is equivalent to shard keys. TcaplusDB tables are split and stored across multiple storage nodes. The splittablekey must be one of the primary key fields, and a good splittablekey should be highly decentralized, meaning the range of values is large. It is recommended to use a string type.
- The desc attribute contains the description of the current element.
- The entry element defines a field, and supported value types include int32, string, char, int64, double, short, etc.
- The index element defines an index that must include a splittable key. Since the primary key can be used to query the table, the index should not be the same as the primary key attributes.

Sample Code for Table Description File in TDR

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>

<metalib name="tcaplus_tb" tagsetversion="1" version="1">

  <!-- generic_table users, store the user' information -->
  <!-- an user may has many roles -->
  <struct name="users" version="1" primarykey="user_id,username,role_id" splittablekey="user_id"
desc="user table">
    <entry name="user_id" type="uint64" desc="user id"/>
    <entry name="username" type="string" size="64" desc="login username"/>
    <entry name="role_id" type="int32" desc="a user can have multiple roles"/>

    <entry name="level" type="int32" defaultvalue="1" desc="role's level"/>
    <entry name="role_name" type="string" size="1024" desc="role's name"/>
    <entry name="last_login_time" type="string" size="64" defaultvalue="" desc="user login timestamp"/>
    <entry name="last_logout_time" type="string" size="64" defaultvalue="" desc="user logout timestamp"/>

    <index name="index1" column="user_id"/>
```

```
  </struct>

  <!-- list_table mails, store the role's mails -->
  <struct name="mails" version="1" primarykey="user_id,role_id" desc="mail table">
    <entry name="user_id" type="uint64" desc="user id"/>
    <entry name="role_id" type="int32" desc="a user may has many roles"/>

    <entry name="text" type="string" size="2048" desc="mail text"/>
    <entry name="send_time" type="string" size="64" defaultvalue="" desc="timestamp of the mail sent"/>
    <entry name="read_time" type="string" size="64" defaultvalue="" desc="timestamp of the mall read"/>
  </struct>

</metalib>
```

Additionally, you can use union to create nested types.
The union element contains a collection of primitive types, such as integers and strings. The union can be referenced as a custom-defined type.
The macro Tag is used for definition constants.

```
<macro name="DB_MAX_USER_MSG_LEN" value="301" desc="Max length of the message that user can define"/>
 <union name="DBPlayerMsg" version="1" desc="DB Player message">
   <entry name="SysMsgID"     type="uint8"         desc="Message ID" />
   <entry name="UsrMsg"       type="string"        size="DB_MAX_USER_MSG_LEN"   desc="player created
message" />
 </union>
```

Creating a Cluster

Last updated: 2024-10-15 17:14:43

Overview

This document describes how to create a TcaplusDB cluster in the console.

> **Note:**
> For TcaplusDB purchases or inquiries, please submit a ticket to contact the Tencent Cloud team for business negotiations to confirm your requirements.

Prerequisites

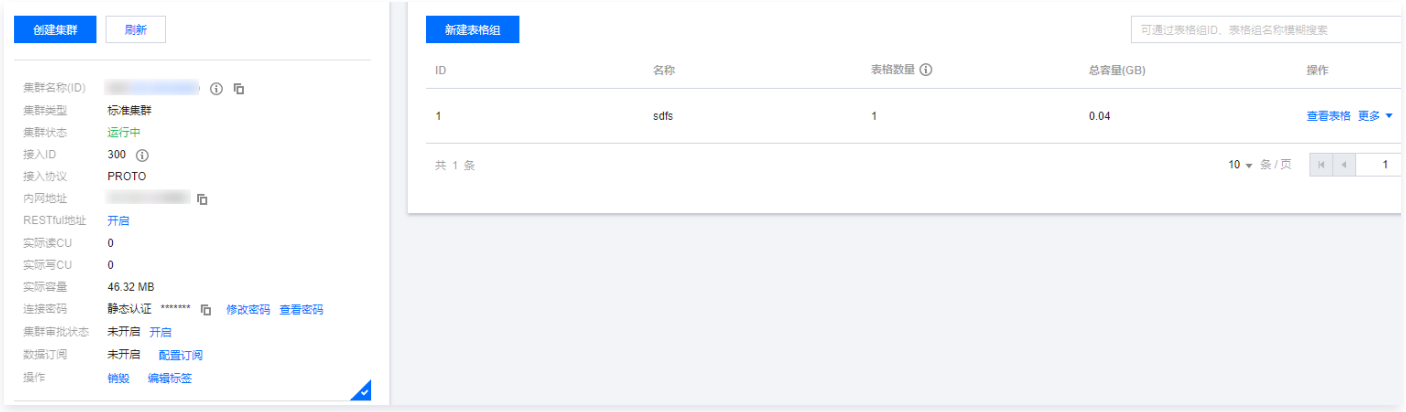
You have registered a Tencent Cloud account and completed real-name authentication.

Directions

1. Log in to the TcaplusDB Console, select **Cluster List** on the left sidebar, and click **Create Cluster**.
2. On the displayed new cluster creation dialog, specify the cluster information and click **Buy Now**.

- **Network**: Select a VPC and a subnet. You can select one VPC and one subnet only when creating a cluster, and this cannot be modified once created.
- **Connection Protocol**: Select a TcaplusDB cluster connection protocol (data description protocol).
- **Cluster Name**: The cluster name must be unique under the account and can contain 1-32 characters. You can rename the cluster at any time.
- **Access Password**: Set the cluster access password with the following requirements:
  - Length [8-64] characters; it is recommended to use a password longer than 12 characters.
  - Cannot start with "/"
  - At least contain three types:
  - Lowercase letters (a - z)
  - Uppercase letters (A - Z)
  - Numbers (0 - 9)
  - Special symbols `\~!@#$%^&*_-+=\|(){}[]:;'<>,.?/`
- **Table Group Name, Table Group ID**: A default table group will be created for the cluster. You can specify its name and Table Group ID, which can be generated automatically or set manually.

3. Go back to the cluster list to view the created cluster and table group. Each cluster will be assigned a unique cluster ID.



Creating Table Group

Last updated: 2024-10-15 17:15:15

Overview

This document describes how to create a table group in the TcaplusDB console.

Prerequisites

You have created a TcaplusDB cluster.

Directions

1. Log in to TcaplusDB Console, select **Cluster List** from the left navigation menu, choose the cluster where you want to create a table group, and click **Create Table Group**.

2. In the pop-up window, specify table group configurations, and click **Create Table Group**. After that, you will be prompted that the creation has been successful.
   - **Select Cluster**: You can switch to another cluster.
   - **Table Group Name**: The name must contain 4-64 characters.
3. Return to the table group list to view information such as the table group's name, table group ID, number of tables, and total capacity.

Creating Table

Last updated: 2024-10-15 17:15:39

Overview

This document describes how to create a table in the TcaplusDB Console.

Prerequisites

- You have created a TcaplusDB cluster and table group.
- You have prepared the table file based on the table description file example.

Directions

1. Log in to the TcaplusDB Console, select **Table List** on the left sidebar, and click **Create Table**.
2. On the create table page, configure the table information.
- **Cluster, Table Group**: Select the target cluster and table group.
- **Table File**: Upload from local or select the table definition file from the previously uploaded files. Users can choose both new and historical files to create the table but cannot upload or select files with the same name. Refer to the sample table file for the ProtoBuf protocol game_players.proto.
- **Remarks**: Enter table remarks.

3. Click **Next**, the system will verify the selected table definition file
- If verification fails, an error will be returned. Please modify the file according to the error prompt and re-upload.
- If verification passes, the table meta information defined in the file will be displayed. Please proceed to click Next.
4. On the table configuration page, check the tables to be created, input capacity, reserved read, and reserved write parameters. The system will automatically calculate the daily cost price of the table.
5. After confirming the table information is correct, click **Create**, and the system will return a success prompt.

Get connection information

Last updated: 2024-10-15 17:16:02

This document describes how to get connection information from the TcaplusDB console.
On the Cluster List page, you can directly view partial information about the cluster. The Access ID, connection password, intranet address, and intranet port information on the cluster overview page are crucial parameters when using the client to connect to TcaplusDB.

- Access ID: The access ID of the cluster.
- Connection Password: The password information for the application.
- Intranet Address: The TcaplusDB connection address.
- Intranet Port: The TcaplusDB connection port.

Search for the required CAM policy as needed, and click to complete policy association.

Accessing TcaplusDB

Last updated: 2024-10-15 17:16:37

TcaplusDB can be accessed and read through multiple ways, such as client tools, SDK toolkits in various programming languages, and RESTful API.

Access TcaplusDB through client tool

Supports accessing TcaplusDB data through the Linux client tool tcaplus_client.

Accessing TcaplusDB Through SDK for C++ API

Supports accessing TcaplusDB data through SDK tools with C++ language.

Access TcaplusDB through RESTful API

TcaplusDB data can be accessed via the RestFul interface. For details, refer to RESTFul API documentation.

- Refer to Python API documentation for accessing TcaplusDB data using Python language via the RESTful interface.
- Refer to PHP API documentation for accessing TcaplusDB data using PHP language via the RESTful interface.
- Refer to Go API documentation for accessing TcaplusDB data using Go language via the RESTful interface.
- Refer to Java API documentation for accessing TcaplusDB data using Java language via the RESTful interface.

External access to TcaplusDB

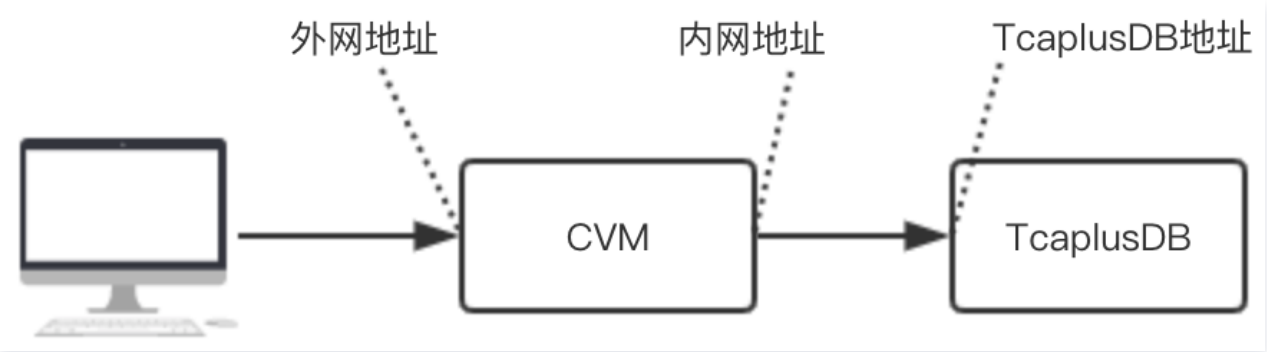
Last updated: 2024-10-15 17:17:02

Overview

TcaplusDB does not currently support external access. You can achieve external access by using a CVM with an external IP to perform port forwarding.

> **Note**
> Because iptables-based forwarding may be unstable, we recommend that you do not access instances over the public network in the production environment.

Search for the required CAM policy as needed, and click to complete policy association.



Directions

1. Log in to CVM and enable the CVM IP forwarding feature.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

2. Configure forwarding rules. The following example shows forwarding access from 26.xx.x.2:6379 to the internal address 10.0.0.5:6379 of TcaplusDB.

> **Note**
> In this example, the external address of the CVM is 26.xx.x.2:6379, and the internal address of TcaplusDB is 10.0.0.5:6379.

```
iptables -t nat -A PREROUTING -p tcp --dport 6379 -j DNAT --to-destination 10.0.0.5:6379
iptables -t nat -A POSTROUTING -d 10.0.0.5 -p tcp --dport 6379 -j MASQUERADE
```

3. Configure CVM security groups to allow access to the external port of the CVM. It is recommended to configure security group rules to allow only the necessary source addresses.
4. You can connect to the internal TcaplusDB via 26.xx.xx.2:6379.