Alibaba Cloud ApsaraDB for Cassandra

Quick start

Issue: 20200526

MORE THAN JUST CLOUD | Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Document conventions

Style	Description	Example
	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 **Danger:** Resetting will result in the loss of user configuration data.
	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	 **Notice:** If the weight is set to 0, the server no longer receives new requests.
	A note indicates supplemental instructions, best practices, tips, and other content.	 **Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings** > **Network** > **Set network type**.
Bold	Bold formatting is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands.	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`

Style	Description	Example
{} or {a|b}	This format is used for a required value, where only one item can be selected.	switch {active|stand}

Contents

1 Use cqlsh to manage an ApsaraDB for Cassandra instance

cqlsh is a command line shell used to interact with Cassandra based on Cassandra Query Language (CQL).

You can download the latest version of Cassandra from the official Apache Cassandra website and decompress the downloaded software package to install Cassandra.

```
$ wget http://mirror.bit.edu.cn/apache/cassandra/3.11.4/apache-cassandra-3.11.4-bin
.tar.gz
$ tar -zxf apache-cassandra-3.11.4-bin.tar.gz
$ cd apache-cassandra-3.11.4
```

Start cqlsh

Log on to the ApsaraDB for Cassandra console, find your ApsaraDB for Cassandra instance, and obtain the endpoint and port of the node to which you want to connect. Then, run the following command to connect to the node:

```
bin/cqlsh $host $port -u $username -p $password
```

If you need to connect to a node frequently, we recommend that you save its endpoint and port to the $CQLSH_HOST and $CQLSH_PORT environment variables. For information about the parameters supported by cqlsh, run the bin/cqlsh -help command.

Common CQL statements

You can run the HELP or ? command in cqlsh to view all CQL statements that can be used to manage ApsaraDB for Cassandra instances.

```
cqlsh> HELP

Documented shell commands:
==========================
CAPTURE  CLS        COPY  DESCRIBE  EXPAND  LOGIN   SERIAL  SOURCE   UNICODE
CLEAR    CONSISTENCY  DESC  EXIT     HELP    PAGING  SHOW    TRACING

CQL help topics:
================
AGGREGATES            CREATE_KEYSPACE         DROP_TRIGGER     TEXT
ALTER_KEYSPACE        CREATE_MATERIALIZED_VIEW  DROP_TYPE       TIME
ALTER_MATERIALIZED_VIEW  CREATE_ROLE           DROP_USER       TIMESTAMP
ALTER_TABLE           CREATE_TABLE            FUNCTIONS       TRUNCATE
ALTER_TYPE            CREATE_TRIGGER          GRANT           TYPES
ALTER_USER            CREATE_TYPE             INSERT          UPDATE
APPLY                 CREATE_USER             INSERT_JSON     USE
```

```
ASCII               DATE                INT             UUID
BATCH               DELETE              JSON
BEGIN               DROP_AGGREGATE      KEYWORDS
BLOB                DROP_COLUMNFAMILY   LIST_PERMISSIONS
BOOLEAN             DROP_FUNCTION       LIST_ROLES
COUNTER             DROP_INDEX          LIST_USERS
CREATE_AGGREGATE    DROP_KEYSPACE       PERMISSIONS
CREATE_COLUMNFAMILY DROP_MATERIALIZED_VIEW REVOKE
CREATE_FUNCTION     DROP_ROLE           SELECT
CREATE_INDEX        DROP_TABLE          SELECT_JSON
```

If you want to know how to execute a specific CQL statement, run the HELP command on that statement. Some CQL statements do not allow you to specify parameters. If you execute such a CQL statement, the system returns the setting queried by that statement. Such CQL statements include CONSISTENCY, EXPAND, and PAGING. Examples:

```
cqlsh> CONSISTENCY
Current consistency level is ONE.
cqlsh> EXPAND
Expanded output is currently disabled. Use EXPAND ON to enable.
cqlsh> PAGING
Query paging is currently enabled. Use PAGING OFF to disable
Page size: 100
```

View environment variables

You can execute the DESCRIBE statement to view the values of environment variables used on an ApsaraDB for Cassandra instance. Example:

```
cqlsh> DESCRIBE CLUSTER;

Cluster: Test Cluster
Partitioner: Murmur3Partitioner
```

The DESCRIBE CLUSTER statement queries the name and partitioner of the ApsaraDB for Cassandra instance. You can choose one of the following four partitioners: RandomPartitioner, Murmur3Partitioner, OrderPreservingPartitioner, or ByteOrderedPartitioner. In versions earlier than Cassandra 1.2, the default partitioner is RandomPartitioner. From Cassandra 1.2 onwards, the default partitioner is Murmur3Partitioner.

To query the keyspaces available on an ApsaraDB for Cassandra instance, execute the following statement:

```
cqlsh> DESCRIBE KEYSPACES;
system_traces system_schema system_auth system system_distributed
```

The system returns all keyspaces provided with the ApsaraDB for Cassandra instance and the keyspaces you have created.

To query the versions of cqlsh, Cassandra, and protocol, execute the following statement:

```
cqlsh> SHOW VERSION;
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
```

Create a keyspace

Keyspaces on an ApsaraDB for Cassandra instance are similar to databases on an ApsaraDB for RDS instance. One keyspace contains one or more tables or column families. If you start cqlsh and do not specify a keyspace, the cqlsh> command prompt is displayed, after which you can execute the CREATE KEYSPACE statement to create a keyspace. Example :

```
cqlsh> CREATE KEYSPACE test_keyspace WITH replication = {'class': 'SimpleStrategy', '
replication_factor': 1};
cqlsh>
```

In this example, you create a keyspace named test_keyspace and set the replication mode to SimpleStrategy. In addition, the ApsaraDB for Cassandra instance used for test consists of only one node. Therefore, set the replication factor to 1. However, if the ApsaraDB for Cassandra instance is in a production environment, we recommend that you set the replication factor to 3.

After you create the keyspace, you can execute the DESCRIBE KEYSPACE statement to query it. Example:

```
cqlsh> DESCRIBE KEYSPACE  test_keyspace;

CREATE KEYSPACE test_keyspace WITH replication = {'class': 'SimpleStrategy', 'replicatio
n_factor': '1'}  AND durable_writes = true;
```

You can also execute the USE statement to switch to the keyspace. Example:

```
cqlsh> USE test_keyspace;
cqlsh:test_keyspace>
```

Create a table

To create a table, execute the following statement:

```
cqlsh> use test_keyspace;
cqlsh:test_keyspace> CREATE TABLE test_user (first_name text , last_name text, PRIMARY
KEY (first_name)) ;
```

In this example, you create a table named test_user in the test_keyspace keyspace. The table contains two fields in the TEXT format: first_name and last_name. The first_name field is the primary key of the table. You can also execute the following statement to create the test_user table in the test_keyspace keyspace:

```
cqlsh> CREATE TABLE test_keyspace.test_user(first_name text , last_name text, PRIMARY KEY (first_name)) ;
```

To query the SQL statements you executed to create the test_user table in the test_keyspace keyspace, execute the following statement:

```
cqlsh:test_keyspace> DESCRIBE TABLE test_user;

CREATE TABLE test_keyspace.test_user (
    first_name text PRIMARY KEY,
    last_name text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTiered
CompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.
compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';

cqlsh:test_keyspace>
```

The system returns all CQL statements you executed to create the test_user table in the test_keyspace, along with the user-defined and default settings.

Read and write data

You can execute the INSERT INTO statement to insert data. Examples:

```
cqlsh:test_keyspace> INSERT INTO test_user (first_name, last_name) VALUES ('test', '
Hadoop');
cqlsh:test_keyspace> INSERT INTO test_user (first_name, last_name) VALUES ('Zhang', '
San');
cqlsh:test_keyspace> INSERT INTO test_user (first_name) VALUES ('Wu');
```

The preceding statements insert three data records into the test_user table. In the last data record, the first_name field is specified with a key and the last_name field is not specified.

You can execute the SELECT COUNT statement to check whether the data records are inserted. Example:

```
cqlsh:test_keyspace> SELECT COUNT(*) FROM test_user;

 count
-------
     3

(1 rows)

Warnings :
Aggregation query used without partition key
```

The return result shows that the data records are inserted. You can also execute the following statement to query the data records you insert:

```
cqlsh:test_keyspace> SELECT * FROM test_user;

 first_name | last_name
------------+-----------
      test |    Hadoop
        Wu |      null
     Zhang |       San

(3 rows)
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name='test';

 first_name | last_name
------------+-----------
      test |    Hadoop

(1 rows)
```

A null value is returned for the last_name field associated with the first_name field whose key is Wu. In Cassandra, a null value indicates that the column specified by the field does not have data, and a column without data does not occupy space in the storage system. However, in common relational databases, a column occupies storage space even if it does not have data.

Delete a column or row

You can use the DELETE statement to delete one or more columns. To delete the last_name column, execute the following statements:

```
cqlsh:test_keyspace> DELETE last_name FROM test_user WHERE first_name='test';
```

```
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name='test';

 first_name | last_name
------------+-----------
       test |      null

(1 rows)
```

The column specified by the last_name field is deleted.

You can also use the DELETE statement to delete a row. Example:

```
cqlsh:test_keyspace> DELETE FROM test_user WHERE first_name='test';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name='test';

 first_name | last_name
------------+-----------

(0 rows)
cqlsh:test_keyspace>
```

The data record whose key is test is deleted.

The INSERT and UPDATE statements used together are equal to the UPSERT statement. If the key associated with the new data record you want to insert already exists, the system does not update the existing data record with the same key, but inserts the new data record and deletes the existing one.

```
cqlsh:test_keyspace> INSERT INTO test_user (first_name, last_name) VALUES ('Wu', 'Shi');
cqlsh:test_keyspace> SELECT * FROM test_user;

 first_name | last_name
------------+-----------
         Wu |       Shi
      Zhang |      San

(2 rows)
```

A value other than null is returned for the last_name field whose key is Wu.

If you use the UPDATE statement to update a data record that does not exist, the system inserts the data record. Example:

```
cqlsh:test_keyspace> SELECT * FROM test_user;

 first_name | last_name
------------+-----------
         Wu |       Shi
      Zhang |      San

(2 rows)
cqlsh:test_keyspace> UPDATE test_user SET last_name = 'Si' WHERE first_name = 'Li';
cqlsh:test_keyspace> SELECT * FROM test_user;
```

```
 first_name | last_name
------------+-----------
         Wu |      Shi
      Zhang |       San
         Li |        Si

(3 rows)
cqlsh:test_keyspace>
```

A data record whose key is Li is inserted into the table, but this data record does not exist before the update.

Clear or delete a table

You can execute the TRUNCATE or DROP TABLE statement to clear or delete a table.

Examples:

```
cqlsh:test_keyspace> TRUNCATE test_user;
cqlsh:test_keyspace> DROP TABLE test_user;
```

2 Cassandra data types

As with other languages, Cassandra Query Language (CQL) supports a flexible set of data types, including primitive data types, collections, and user-defined types (UDTs). This topic describes the data types that CQL supports.

The numeric data types supported by CQL include integer and floating-point numbers. These types are similar to standard types in Java. Specifically, CQL supports the following numeric data types:

- int: a 32-bit signed integer, as in Java.
- bigint: a 64-bit long integer, which is equivalent to long in Java.
- smallint: a 16-bit signed integer, which is equivalent to short in Java. This data type was introduced in Apache Cassandra 2.2.
- tinyint: an 8-bit signed integer, as in Java. This data type was introduced in Apache Cassandra 2.2.
- varint: a variable-precision signed integer, which is equivalent to java.math.BigInteger.
- float: a 32-bit IEEE-754 floating point, as in Java.
- double: a 64-bit IEEE-754 floating point, as in Java.
- decimal: a variable-precision decimal, which is equivalent to java.math.BigDecimal.

Textual data types

CQL provides the following data types for representing text:

- text or varchar: a UTF-8 encoded character string, which is commonly used in CQL.
- ascii: an ASCII character string.

Time and identity data types

- timestamp: The time can be encoded as a 64-bit signed integer, but it is typically much more useful to enter a timestamp by using one of several supported ISO 8601 date formats. We recommend that you always provide time zones for timestamps rather than relying on the time zone configuration of the operating system.
- date and time: Apache Cassandra 2.1 and earlier only had the timestamp type to represent a date and time. The 2.2 release introduced date and time types that allowed dates and time to be represented independently. As with timestamp, these types support ISO 8601 formats.

- uuid: A universally unique identifier (UUID) is a 128-bit value in which the bits conform to one of several types, of which the most commonly used are known as Type 1 and Type 4 . The CQL uuid type is a Type 4 UUID, which is based entirely on random numbers. UUIDs are typically represented as dash-separated sequences of hexadecimal digits, such as ab7c46 ac-c194-4c71-bb03-0f64986f3daa. The uuid type is often used as a surrogate key, either by itself or in combination with other values. Because UUIDs are of a finite length, they are not absolutely guaranteed to be unique. You can use the uuid() function in CQL to obtain a Type 4 UUID value.
- timeuuid: This is a Type 1 UUID, which is based on the MAC address of the computer, the system time, and a sequence number used to prevent duplicates. CQL provides several convenience functions for interacting with the timeuuid type, such as now(), dateOf(), and unixTimestampOf(). The availability of these convenience functions is one reason why timeuuid tends to be used more frequently than uuid.

Collection data types

Collection data types can store a collection of data. The elements stored in the set data type are unordered, but CQL returns the elements in sorted order. Sets can contain the data types mentioned earlier as well as user-defined types and even other collections.

The following example demonstrates how to use the set data type to store email informatio n:

```
cqlsh:test_keyspace> CREATE TABLE test_user (first_name text , last_name text,emails set
<text>, PRIMARY KEY (first_name)) ;
cqlsh:test_keyspace> INSERT INTO test_user (first_name, last_name,emails) VALUES ('Wu',
 'Shi',{'iteblog@iteblog.com'});
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails                 | last_name
------------+------------------------+-----------
         Wu | {'iteblog@iteblog.com'} |       Shi

(1 rows)
```

In the preceding statements, an email address is added for the user whose first_name is Wu . If you want to add another email address, use the following syntax:

```
cqlsh:test_keyspace> UPDATE test_user SET emails = emails + {'cassandra@iteblog.com' }
WHERE first_name = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails                                        | last_name
------------+-----------------------------------------------+-----------
         Wu | {'cassandra@iteblog.com', 'iteblog@iteblog.com'} |       Shi

(1 rows)
```

Two email addresses are added for the user whose first_name is Wu. If you want to delete an email address, use the following syntax:

```
cqlsh:test_keyspace> UPDATE test_user SET emails = emails - {'cassandra@iteblog.com'}
WHERE first_name = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails                 | last_name
------------+------------------------+-----------
       Wu | {'iteblog@iteblog.com'} |      Shi

(1 rows)
cqlsh:test_keyspace> UPDATE test_user SET emails ={} WHERE first_name = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails | last_name
------------+--------+-----------
       Wu |   null |      Shi

(1 rows)
```

In the preceding statements, SET emails = emails - {'cassandra@iteblog.com'} is used to remove an email address from the email list and SET emails ={} is used to clear all email information of the user.

list

The list data type contains an ordered list of elements. By default, the values are stored in order of insertion. The following example demonstrates how to add information such as phone numbers to the test_user table:

```
cqlsh:test_keyspace> ALTER TABLE test_user ADD phone list<text>;
cqlsh:test_keyspace> UPDATE test_user SET phone = ['13112345678' ] WHERE first_name =
 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-----------------
       Wu |   null |      Shi | ['13112345678']

(1 rows)
```

In the preceding statements, a phone number is added for the user whose first_name is Wu. If you want to add another phone number, use the following syntax similar to that of set:

```
cqlsh:test_keyspace> UPDATE test_user SET phone = phone + ['15511112222' ] WHERE
first_name = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+--------------------------------
```

```
      Wu |   null |      Shi | ['13112345678', '15511112222']

(1 rows)
```

In the output, the newly added phone number appears at the end of the list. You can use the following statements to prepend a phone number to the front of the list:

```
cqlsh:test_keyspace> UPDATE test_user SET phone = ['13344448888' ] + phone WHERE
first_name = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-------------------------------------------------
      Wu |   null |      Shi | ['13344448888', '13112345678', '15511112222']

(1 rows)
```

You can modify an individual item in the list when you reference it by its index:

```
cqlsh:test_keyspace> UPDATE test_user SET phone[1] = '18888888888' WHERE first_name
 = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+-------------------------------------------------
      Wu |   null |      Shi | ['13344448888', '18888888888', '15511112222']

(1 rows)
```

The element with an index of 1 is modified. You can also delete a specific item by using its index:

```
cqlsh:test_keyspace> DELETE phone[2] from test_user WHERE first_name = 'Wu';
cqlsh:test_keyspace> SELECT * FROM test_user WHERE first_name = 'Wu';

 first_name | emails | last_name | phone
------------+--------+-----------+--------------------------------
      Wu |   null |      Shi | ['13344448888', '18888888888']

(1 rows)
```

You can also use SET phone_numbers = phone_numbers - ['13344448888'] to delete an element.

Map

The map data type contains a collection of key-value pairs. The keys and values can be of any types except counter. Example:

```
cqlsh:test_keyspace> ALTER TABLE test_user ADD login_sessions map<timeuuid, int>;
cqlsh:test_keyspace> UPDATE test_user SET login_sessions = {now(): 13, now(): 18}
WHERE first_name = 'Wu';
```

```
cqlsh:test_keyspace> SELECT first_name, login_sessions FROM test_user WHERE
first_name = 'Wu';

 first_name | login_sessions
------------
+-----------------------------------------------------------------------------------
       Wu | {1cc61ff0-5f8b-11e9-ac3a-5336cd8118f6: 13, 1cc61ff1-5f8b-11e9-ac3a-
5336cd8118f6: 18}

(1 rows)
```

Other simple data types

- boolean: The value is either true or false. CQL is case insensitive in accepting these values but returns True or False.
- blob: A binary large object (blob) is a colloquial computing term for an arbitrary array of bytes. The CQL blob type is useful for storing media or other binary file types. Cassandra does not validate or examine the bytes in a blob. In Cassandra, blobs are represented as hexadecimal digits. If you want to encode arbitrary textual data into a blob, you can use the textAsBlob() function.
- inet: This type represents IPv4 or IPv6 addresses. cqlsh accepts any valid formats for defining IPv4 addresses, including dotted or non-dotted representations containing decimal, octal, or hexadecimal values. However, the values are represented by using the dotted decimal format in CQL output, such as 1.1.1.1.
- counter: The counter data type is a 64-bit signed integer, whose value cannot be set directly, but only incremented or decremented. The counter type has some special restrictions. It cannot be used as part of a primary key. If a counter is used, all of the columns other than primary keys must be counters.

UDTs

If the built-in data types in Cassandra do not meet your requirements, you can use UDTs. For example, if you want to use a column to store the address information of a user, you need to obtain information such as the zip code and street. Using a text column to store these values may not meet your requirements. In this case, you can define a UDT. Example:

```
cqlsh:test_keyspace> CREATE TYPE address (
                ... street text,
                ... city text,
                ... state text,
                ... zip_code int);
```

The preceding statement defines the address data type. Note that a UDT is scoped by the keyspace in which it is defined. This indicates that the address data type can only be used

in test_keyspace. If you execute DESCRIBE KEYSPACE test_keyspace, the output shows that the address data type is part of test_keyspace. The following example demonstrates how to use the defined address type:

```
cqlsh:test_keyspace> ALTER TABLE test_user ADD addresses map<text, frozen<address>>;
cqlsh:test_keyspace> UPDATE test_user SET addresses = addresses + {'home': { street: '
shangdi 9', city: 'Beijing', state: 'Beijing', zip_code: 100080} } WHERE first_name = 'Wu';
cqlsh:test_keyspace> SELECT first_name, addresses FROM test_user WHERE first_name = '
Wu';

 first_name | addresses
------------
+-------------------------------------------------------------------------------------
       Wu | {'home': {street: 'shangdi 9', city: 'Beijing', state: 'Beijing', zip_code: 100080}}

(1 rows)
```

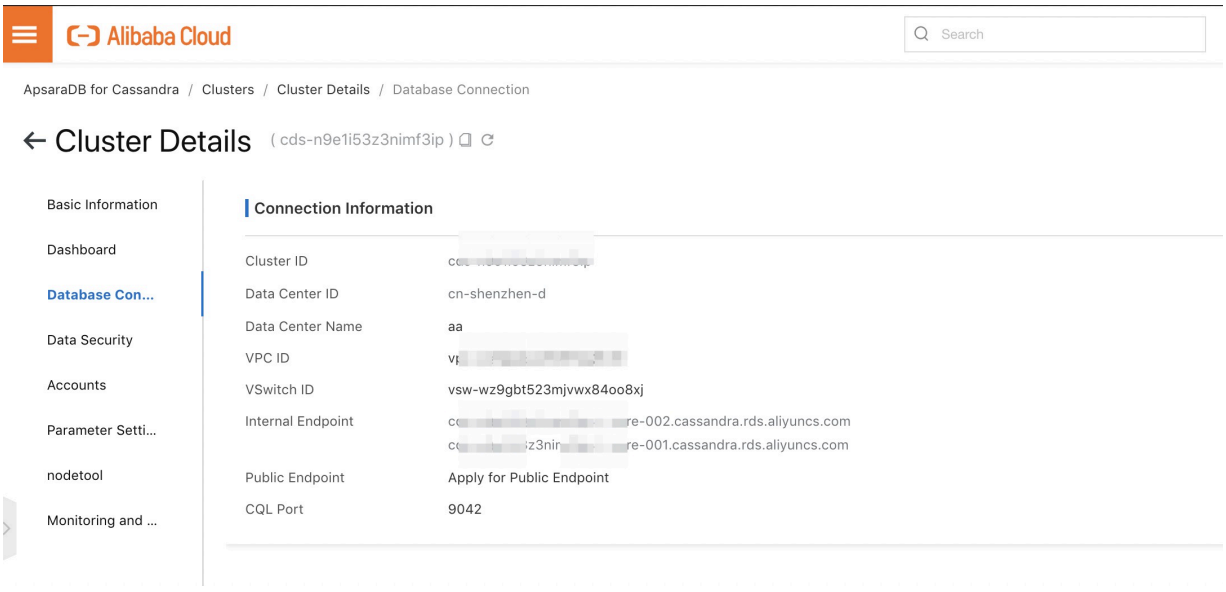
3 Use multi-language SDKs to access an ApsaraDB for Cassandra instance over the Internet and internal network

You have learned how to use cqlsh to access an ApsaraDB for Cassandra instance. However, in practice, you may need to write code to access an ApsaraDB for Cassandra instance, rather than merely connecting to a node and typing some CQL statements like you would do in cqlsh. You may need to connect to multiple Cassandra nodes, evenly distribute requests to each Cassandra node, and manage connection pools. You do not have to worry about this because the community versions of SDKs have already implemented these features. The following sections describe how to use the SDKs to access an ApsaraDB for Cassandra instance.

Preparations

1. Obtain the endpoint

Log on to the ApsaraDB for Cassandra console. Click a cluster ID to go to the Cluster Details page of the cluster. In the left-side navigation pane, click Database Connection.



For security considerations, Internet connection is disabled by default. To obtain the public endpoint, enable Internet connection as shown in the preceding figure. Endpoints in the following formats are obtained:

```
cds-xxxxxxxx-core-003.cassandra.rds.aliyuncs.com
cds-xxxxxxxx-core-002.cassandra.rds.aliyuncs.com
```

The number of endpoints varies with the cluster size. Use as many endpoints as possible. This helps prevent failure to connect to the cluster due to the failure of a single node. The methods for accessing an ApsaraDB for Cassandra instance over the Internet and internal network are almost identical except that different endpoints are used. The following examples use Java and Python code.

2. Add IP addresses to a whitelist

Before you access an ApsaraDB for Cassandra instance, ensure that you have added the IP addresses of clients to the instance whitelist. For more information, visit Configure a whitelist.

Multi-language sample code

1. Java-based access

1.1 Add a Maven dependency

```
<dependency>
   <groupId>com.taobao.pandora</groupId>
   <artifactId>odps-sdk-core</artifactId>
   <version>3.7.1</version>
</dependency>
```

This reference introduces some public libraries. To avoid unnecessary trouble caused by dependency conflicts, test the dependency in a new project first.

1.2 Write Java code for the access

```
import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.PlainTextAuthProvider;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Session;

public class Demo {

  public static void main(String[] args) {

    // Enter the public or internal endpoints of the database. You can enter as many endpoints as provided by the console.
    // In fact, the SDK will only connect to the first accessible endpoint and establish a control connection. You can enter multiple endpoints to avoid database connection failure caused by the failure of a single node.
    // Ignore the sequence of the endpoints, because the SDK may rearrange the sequence to prevent different clients from connecting to the same endpoint.
    // You must enter only public endpoints or only internal endpoints.
    String[] contactPoints = new String[]{
      "cds-xxxxxxxx-core-003.cassandra.rds.aliyuncs.com",
      "cds-xxxxxxxx-core-002.cassandra.rds.aliyuncs.com"
    };
```

```
    Cluster cluster = Cluster.builder()
      .addContactPoints(contactPoints)
      // Enter the account name and password. If you forget the password, you can reset it on the Accounts page.
      .withAuthProvider(new PlainTextAuthProvider("cassandra", "123456"))
      // If you access the Cassandra cluster over the Internet, you must append @public to the account name to switch to full Internet link.
      // Otherwise, you cannot connect to all internal nodes over the Internet, and exceptions or delays will occur, affecting local development and debugging.
      // Automatic identification of network links will be supported in the future, eliminating the need to manually add @public. For more information, see changelogs on the official website.
      //.withAuthProvider(new PlainTextAuthProvider("cassandra@public", "123456"))
      .build();

    // Initialize the cluster. A control connection will be established at this time. This step can be ignored, because the init method is automatically called when a session is established.
    cluster.init();

    // Connect to the cluster. A persistent connection pool will be established for each Cassandra node.
    // Therefore, it is burdensome to create one session for each request. We recommend that you create several sessions in advance for each process.
    // As a general rule, one session is sufficient. You can make adjustments based on your actual needs. For example, you can manage the read and write sessions separately.
    Session session = cluster.connect();

    // Query the permission-related table to view the number of created roles.
    // This is the system table. By default, only the superuser account cassandra has the SELECT permission.
    // If you use another account for testing, you can switch to another table or grant the SELECT permission to the account.
    ResultSet res = session.execute("SELECT * FROM system_auth.roles");

    // ResultSet implements the Iterable interface to display each line of information in the console.
    res.forEach(System.out::println);

    // Close the session.
    sqlSession.close();

    // Shut down the cluster.
    writer.close();
  }

}
```

2. Python-based access

2.1 Install the SDK library

```
# Install the specified version. Python 3.x is recommended.
pip install cassandra-driver==3.19.0
# Install the latest version.
pip install cassandra-driver
# https://pypi.org/project/cassandra-driver/#history
```

2.2 Write Python code for the access

```
#! /usr/bin/env python
# -*- coding: UTF-8 -*-

import logging
import sys
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider

logging.basicConfig(stream=sys.stdout, level=logging.INFO)


cluster = Cluster(
    # Enter the public or internal endpoints of the database. You can enter as many endpoints as provided by the console.
    # In fact, the SDK will only connect to the first accessible endpoint and establish a control connection. You can enter multiple endpoints to avoid database connection failure caused by the failure of a single node.
    # Ignore the sequence of the endpoints, because the SDK may rearrange the sequence to prevent different clients from connecting to the same endpoint.
    # You must enter only public endpoints or only internal endpoints.
    contact_points=["cds-xxxxxxxx-core-003.cassandra.rds.aliyuncs.com",
                    "cds-xxxxxxxx-core-002.cassandra.rds.aliyuncs.com"],
    # Enter the account name and password. If you forget the password, you can reset it on the Accounts page.
    auth_provider=PlainTextAuthProvider("cassandra", "123456"))
    # If you access the Cassandra cluster over the Internet, you must append @public to the account name to switch to full Internet link.
    # Otherwise, you cannot connect to all internal nodes over the Internet, and exceptions or delays will occur, affecting local development and debugging.
    # Automatic identification of network links will be supported in the future, eliminating the need to manually add @public. For more information, see changelogs on the official website.
    # auth_provider=PlainTextAuthProvider("cassandra@public", "123456"))

# Connect to the cluster. A persistent connection pool will be established for each Cassandra node.
# Therefore, it is burdensome to create one session for each request. We recommend that you create several sessions in advance for each process.
# As a general rule, one session is sufficient. You can make adjustments based on your actual needs. For example, you can manage the read and write sessions separately.
session = cluster.connect()

# Query the permission-related table to view the number of created roles.
# This is the system table. By default, only the superuser account cassandra has the SELECT permission.
# If you use another account for testing, you can switch to another table or grant the SELECT permission to the account.
rows = session.execute('SELECT * FROM system_auth.roles')

# Display each line of information in the console.
for row in rows:
    print("# row: {}".format(row))

# Close the session.
session.shutdown()

# Shut down the cluster.
cluster.shutdown()
```

3. Access in other programming languages

The access interfaces for other programming languages are similar to those of Python and Java and can be found in the GitHub community documentation.

- Ruby
- C# and .NET
- Node.js
- PHP
- C++