

Alibaba Cloud

Tablestore
Data channels

Document Version: 20220221

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 because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should 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 this document based on the "status quo", "being defective", and "existing functions" of its products and services. 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 take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, 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 directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
 Danger	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.
 Warning	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.
 Notice	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.
 Note	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 <code>cd /d C:/window</code> command to enter the Windows system folder.
<i>Italic</i>	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	This format is used for a required value, where only one item can be selected.	<code>switch {active stand}</code>

Table of Contents

1.Data import	05
1.1. Synchronize Kafka data to Tablestore	05
1.1.1. Overview	05
1.1.2. Data synchronization to data tables	06
1.1.3. Data synchronization to time series tables	13
1.1.4. Configuration description	18
1.1.5. Error handling	43
1.2. Synchronize data from one table to another table	44
2.Data export	47
2.1. Synchronize data from tablestore to OSS	47
2.1.1. Overview	47
2.1.2. Export full data in script mode	48
2.1.3. Synchronize incremental data in script mode	54
2.2. Synchronize data from tablestore to MaxCompute	59
2.2.1. Export full data in script mode	59

1.Data import

1.1. Synchronize Kafka data to Tablestore

1.1.1. Overview

You can use Tablestore Sink Connector to batch import data in Apache Kafka to a data table or time series table in Tablestore.

Background information

Apache Kafka is a distributed Message Queuing (MSMQ) system. Data systems can use Kafka Connect to import data streams to and export data streams from Apache Kafka.

The Tablestore team has developed Tablestore Sink Connector based on Kafka Connect. Tablestore Sink Connector pulls message records based on the subscribed topics from Apache Kafka in poll mode, parses the message records, and then batch imports the data to Tablestore. Tablestore Sink Connector optimizes the process of importing data and supports custom configurations.

Tablestore is a multi-model data storage service that is developed by Alibaba Cloud. Tablestore can store large amounts of structured data and supports a variety of data models, including the Wide Column model and the TimeSeries model. You can synchronize data from Apache Kafka to a data table or time series table in Tablestore. Data tables are a table type in the Wide Column model and time series tables are a table type in the TimeSeries model. For more information about specific operations, see [Data synchronization to data tables](#) and [Data synchronization to time series tables](#).

Features

Tablestore Sink Connector supports the following features:

- **At-least-once delivery**
Ensures that Kafka message records are delivered from Kafka topics to Tablestore at least once.
- **Data mapping**
Deserializes data in Kafka topics by using Converter. Before you deserialize data by using Converter, you need to modify the `key.converter` and `value.converter` attributes in the worker or connector configurations of Kafka Connect. You can choose the `JsonConverter` that is built in Kafka Connect, a third-party Converter, or a custom Converter.
- **Automatic creation of destination tables in Tablestore**
If the destination table is missing in Tablestore, a destination table can be automatically created based on the primary key columns and attribute column whitelist that you specify. If no attribute column whitelist is specified, all fields in the record values of Kafka message records are used as the attribute columns of the destination table.
- **Error handling policy**
Errors may occur when message records are parsed or written to Tablestore because data is imported in batches. If an error occurs, you can terminate the task or ignore the error. You can also log the message record and the error message in Kafka or Tablestore.

Working mode

Tablestore Sink Connector can work in the standalone or distributed mode. You can select a mode based on your business requirements.


- In the standalone mode, all tasks are executed in a single process. This mode is easy to configure and use. You can use the standalone mode to learn about the features of Tablestore Sink Connector.
- In the distributed mode, all tasks are executed in multiple processes in parallel. This mode can allocate tasks to processes based on the workloads of the processes and provides the fault tolerance capability when tasks are executed. This way, the distributed mode outperforms the standalone mode in stability. We recommend that you use the distributed mode.

1.1.2. Data synchronization to data tables

Tablestore Sink Connector pulls message records based on the subscribed topics from Apache Kafka in poll mode, parses the message records, and then batch imports the data to a data table in Tablestore.

Prerequisites

- Apache Kafka is installed and enabled, and ZooKeeper is enabled. For more information, see [Kafka documentation](#).
- The Tablestore service is activated, and an instance and a data table are created. For more information, see [Use the Wide Column model](#).

 **Note** You can also use Tablestore Sink Connector to automatically create a destination data table. To create this data table, set `auto.create` to `true`.

- An AccessKey pair is obtained. For more information, see [Obtain an AccessKey pair](#).

Step 1: Deploy Tablestore Sink Connector

1. Obtain the Tablestore Sink Connector package by using one of the following methods:
 - Download the source code from [Tablestore Sink Connector source code](#) on GitHub and compile the source code.
 - a. Run the following command to download the source code of Tablestore Sink Connector by using the Git tool:

```
git clone https://github.com/aliyun/kafka-connect-tablestore.git
```

- b. Go to the directory where the source code that you downloaded is stored, and run the following command to package the source code by using Maven:

```
mvn clean package -DskipTests
```

After the compilation is complete, the generated package is stored in the target directory. The `kafka-connect-tablestore-1.0.jar` package is used as an example.

- Download the [kafka-connect-tablestore package](#) that has been compiled.
2. Copy the package to the `$KAFKA_HOME/libs` directory on each node.

Step 2: Start Tablestore Sink Connector

Tablestore Sink Connector can work in the standalone or distributed mode. You can select a mode based on your business requirements.

To use Tablestore Sink Connector in the standalone mode, perform the following steps:

1. Modify the worker configuration file `connect-standalone.properties` and the connector configuration file `connect-tablestore-sink-quickstart.properties` based on your requirements.

- Example on how to modify the worker configuration file `connect-standalone.properties`

The worker configuration file contains configuration items. These items include the Kafka connection parameters, the serialization format, and the frequency at which the offsets are committed. The following sample code is an example that is provided by Apache Kafka on how to modify the worker configuration file. For more information, see [Kafka Connect](#).

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# These are defaults. This file just demonstrates how to override some settings.
bootstrap.servers=localhost:9092
# The converters specify the format of data in Kafka and how to translate it into Con
nect data. Every Connect user will
# need to configure these based on the format they want their data in when loaded fro
m or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting w
ith the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.file.filename=/tmp/connect.offsets
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000
# Set to a list of filesystem paths separated by commas (,) to enable class loading i
solation for plugins
# (connectors, converters, transformations). The list should consist of top level dir
ectories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of
plugins and their dependencies
# Note: symlinks will be followed to discover dependencies or plugins.
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
#plugin.path=
```

- Example on how to modify the connector configuration file `connect-tablestore-sink-quickstart.properties`

The connector configuration file contains configuration items. These items include the connector class, Tablestore connection parameters, and data mapping. For more information, see [Configuration description](#).

```
# Specify the connector name.
name=tablestore-sink
# Specify the connector class.
connector.class=TableStoreSinkConnector
# Specify the maximum number of tasks.
tasks.max=1
# Specify the list of Kafka topics from which data is exported.
topics=test
# Specify values for the following Tablestore connection parameters:
# The endpoint of the Tablestore instance.
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# The AccessKey pair which consists of an AccessKey ID and an AccessKey secret.
tablestore.access.key.id =xxx
tablestore.access.key.secret=xxx
# The name of the Tablestore instance.
tablestore.instance.name=xxx
# Specify the format string for the name of the destination table in Tablestore. <topic> is a placeholder for the topic from which you want to export data. Default value: <topic>.
# Examples:
# If table.name.format=kafka_<topic> is specified, the message records from the topic named test are written to the data table named kafka_test.
# table.name.format=
# Specify the primary key mode. Default value: kafka.
# If the primary key mode is set to kafka, <topic>_<partition> and <offset> are used as the primary key of the Tablestore data table. <topic>_<partition> specifies the Kafka topic and partition, which are separated by an underscore (_). <offset> specifies the offset of the message record in the partition.
# primarykey.mode=
# Specify whether to automatically create a destination table. Default value: false.
auto.create=true
```

2. Go to the `$KAFKA_HOME` directory and run the following command to enable the standalone mode:

```
bin/connect-standalone.sh config/connect-standalone.properties config/connect-tablestore-sink-quickstart.properties
```

To use Tablestore Sink Connector in the distributed mode, perform the following steps:

1. Modify the worker configuration file `connect-distributed.properties` based on your business requirements.

The worker configuration file contains configuration items. These items include the Kafka connection parameters, the serialization format, the frequency at which the offsets are committed, and the topics that store connector information. We recommend that you create the topics in advance. The following sample code is an example that is provided by Apache Kafka on how to modify the worker configuration file. For more information, see [Kafka Connect](#).

- `offset.storage.topic`: specifies the compact topic where connector offsets are stored.
- `config.storage.topic`: specifies the compact topic where connector and task configurations are stored. The number of partitions for the compact topic must be set to 1.
- `status.storage.topic`: specifies the compact topic where the status information about Kafka Connect is stored.

```
##
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
##
# This file contains some of the configurations for the Kafka Connect distributed worke
r. This file is intended
# to be used with the examples, and some settings may differ from those used in a produ
ction system, especially
# the `bootstrap.servers` and those specifying replication factors.
# A list of host/port pairs to use for establishing the initial connection to the Kafka
cluster.
bootstrap.servers=localhost:9092
# unique name for the cluster, used in forming the Connect cluster group. Note that thi
s must not conflict with consumer group IDs
group.id=connect-cluster
# The converters specify the format of data in Kafka and how to translate it into Conne
ct data. Every Connect user will
# need to configure these based on the format they want their data in when loaded from
or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting wit
h the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=true
# Topic to use for storing offsets. This topic should have many partitions and be repli
cated and compacted.
# Kafka Connect will attempt to create the topic automatically when needed, but you can
always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some c
ases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication fa
ctor used, we'd like to be able
# to run this example on a single broker cluster and so here we instead set the replic
```


```

# to run this example on a single-broker cluster and so here we instead set the replica
tion factor to 1.
offset.storage.topic=connect-offsets
offset.storage.replication.factor=1
#offset.storage.partitions=25
# Topic to use for storing connector and task configurations; note that this should be
a single partition, highly replicated,
# and compacted topic. Kafka Connect will attempt to create the topic automatically whe
n needed, but you can always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some c
ases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication fa
ctor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replica
tion factor to 1.
config.storage.topic=connect-configs
config.storage.replication.factor=1
# Topic to use for storing statuses. This topic can have multiple partitions and should
be replicated and compacted.
# Kafka Connect will attempt to create the topic automatically when needed, but you can
always manually create
# the topic before starting Kafka Connect if a specific topic configuration is needed.
# Most users will want to use the built-in default replication factor of 3 or in some c
ases even specify a larger value.
# Since this means there must be at least as many brokers as the maximum replication fa
ctor used, we'd like to be able
# to run this example on a single-broker cluster and so here we instead set the replica
tion factor to 1.
status.storage.topic=connect-status
status.storage.replication.factor=1
#status.storage.partitions=5
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000
# These are provided to inform the user about the presence of the REST host and port co
nfigs
# Hostname & Port for the REST API to listen on. If this is set, it will bind to the in
terface used to listen to requests.
#rest.host.name=
#rest.port=8083
# The Hostname & Port that will be given out to other workers to connect to i.e. URLs t
hat are routable from other servers.
#rest.advertised.host.name=
#rest.advertised.port=
# Set to a list of filesystem paths separated by commas (,) to enable class loading iso
lation for plugins
# (connectors, converters, transformations). The list should consist of top level direc
tories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of p
lugins and their dependencies
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors.

```

```
#plugin.path=/usr/local/share/java:/usr/local/share/java/plugins:/usr/local/share/java/connectors,
#plugin.path=
```

2. Go to the \$KAFKA_HOME directory and run the following command to enable the distributed mode:

 **Notice** You need to start the worker process on each node.

```
bin/connect-distributed.sh config/connect-distributed.properties
```

3. Manage connectors by using the REST API. For more information, see [REST API](#).
 - i. Create a file named connect-tablestore-sink-quickstart.json in the config path. The following sample code provides an example of the content that you need to add to the file.

The connector configuration file specifies the key-value pairs for the configuration items by using strings in the JSON format. These items include the connector class, Tablestore connection parameters, and data mapping. For more information, see [Configuration description](#).

```
{
  "name": "tablestore-sink",
  "config": {
    "connector.class": "TableStoreSinkConnector",
    "tasks.max": "1",
    "topics": "test",
    "tablestore.endpoint": "https://xxx.xxx.ots.aliyuncs.com",
    "tablestore.access.key.id": "xxx",
    "tablestore.access.key.secret": "xxx",
    "tablestore.instance.name": "xxx",
    "table.name.format": "<topic>",
    "primarykey.mode": "kafka",
    "auto.create": "true"
  }
}
```

- ii. Run the following command to start a Tablestore Sink Connector client:

```
curl -i -k -H "Content-type: application/json" -X POST -d @config/connect-tablestore-sink-quickstart.json http://localhost:8083/connectors
```

In the preceding command, `http://localhost:8083/connectors` is the address of the Kafka REST service. Modify the address based on your business requirements.

Step 3: Generate message records

1. Go to the \$KAFKA_HOME directory and run the following command to start a console producer client:

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

The following table describes the parameters that you need to configure to start a console producer client.

Parameter	Example	Description
--broker-list	localhost:9092	The address and port of the broker in the Kafka cluster.
--topic	test	The name of the topic. By default, a topic is automatically created when you start Tablestore Sink Connector. You can also manually create a topic.

2. Write messages to the topic named test.

◦ Messages in a Struct

```
{
  "schema":{
    "type":"struct",
    "fields":[
      {
        "type":"int32",
        "optional":false,
        "field":"id"
      },
      {
        "type":"string",
        "optional":false,
        "field":"product"
      },
      {
        "type":"int64",
        "optional":false,
        "field":"quantity"
      },
      {
        "type":"double",
        "optional":false,
        "field":"price"
      }
    ],
    "optional":false,
    "name":"record"
  },
  "payload":{
    "id":1,
    "product":"foo",
    "quantity":100,
    "price":50
  }
}
```

◦ Messages in a Map

```
{
  "schema": {
    "type": "map",
    "keys": {
      "type": "string",
      "optional": false
    },
    "values": {
      "type": "int32",
      "optional": false
    },
    "optional": false
  },
  "payload": {
    "id": 1
  }
}
```

3. Log on to the [Tablestore console](#) to view data.

A data table named test is automatically created in the Tablestore instance. The following figure shows the data in the data table. Data in the first row is the result of the messages in a Map that are imported and data in the second row is the result of the messages in a Struct that are imported.


<input type="checkbox"/>	Details	topic_partition(Primary Key)	offset(Primary Key)	id	price	product	quantity
<input type="checkbox"/>	Details	test_3	0	AAAAAQ==			
<input type="checkbox"/>	Details	test_34	0	1	50.0	foo	100

1.1.3. Data synchronization to time series tables

You can use the kafka-connect-tablestore package to synchronize data from Apache Kafka to a time series table in Tablestore. This topic describes how to configure data synchronization from Kafka to time series tables in Tablestore.

Prerequisites

- Apache Kafka is installed and enabled, and ZooKeeper is enabled. For more information, see [Kafka documentation](#).
- The Tablestore service is activated, and an instance and a time series table are created. For more information, see [Use the TimeSeries model](#).

 **Note** You can also use Tablestore Sink Connector to automatically create a destination time series table. To create this time series table, set `auto.create` to `true`.

- An AccessKey pair is obtained. For more information, see [Obtain an AccessKey pair](#).

Context

Tablestore can store time series data and supports analytics on time series data. For more information, see [Overview](#).

Step 1: Deploy Tablestore Sink Connector

1. Obtain the Tablestore Sink Connector package by using one of the following methods:
 - Download the source code from [Tablestore Sink Connector source code](#) on GitHub and compile the source code.
 - a. Run the following command to download the source code of Tablestore Sink Connector by using the Git tool:

```
git clone https://github.com/aliyun/kafka-connect-tablestore.git
```

- b. Go to the directory where the source code that you downloaded is stored, and run the following command to package the source code by using Maven:

```
mvn clean package -DskipTests
```


After the compilation is complete, the generated package is stored in the target directory. The kafka-connect-tablestore-1.0.jar package is used as an example.

- Download the [kafka-connect-tablestore package](#) that has been compiled.
2. Copy the package to the \$KAFKA_HOME/libs directory on each node.

Step 2: Start Tablestore Sink Connector

Tablestore Sink Connector can work in the standalone or distributed mode. You can select a mode based on your business requirements.

To write time series data to Tablestore, message records in Kafka must be in the JSON format. Therefore, JsonConverter is required to start Tablestore Sink Connector. You do not need to extract the schema and enter the key, but you must configure the configuration items in connect-standalone.properties or connect-distributed.properties. The following sample code shows how to configure the configuration items.

 **Note** If you enter the key, you must configure key.converter and key.converter.schemas.enable based on the format of the key.

```
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=false
```

This section describes how to synchronize data to a time series table in Tablestore in the standalone mode. The procedure to synchronize data to a time series table in Tablestore in the distributed mode is similar to the procedure to synchronize data to a data table in Tablestore in the distributed mode. However, you need to modify the preceding configuration items in the worker configuration file connect-distributed.properties and modify time series-related configuration items in the connector configuration file connect-tablestore-sink-quickstart.json. For more information, see the configuration procedure in the distributed mode in [Step 2: Start Tablestore Sink Connector](#).

To use Tablestore Sink Connector in the standalone mode, perform the following steps:

1. Modify the worker configuration file connect-standalone.properties and the connector configuration file connect-tablestore-sink-quickstart.properties based on your requirements.
 - Example on how to modify the worker configuration file connect-standalone.properties

The worker configuration file contains configuration items. These items include the Kafka connection parameters, the serialization format, and the frequency at which the offsets are committed. The following sample code is an example that is provided by Apache Kafka on how to modify the worker configuration file. For more information, see [Kafka Connect](#).

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# These are defaults. This file just demonstrates how to override some settings.
bootstrap.servers=localhost:9092
# The converters specify the format of data in Kafka and how to translate it into Con
nect data. Every Connect user will
# need to configure these based on the format they want their data in when loaded fro
m or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting w
ith the converter we want to apply
# it to
key.converter.schemas.enable=true
value.converter.schemas.enable=false
offset.storage.file.filename=/tmp/connect.offsets
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000
# Set to a list of filesystem paths separated by commas (,) to enable class loading i
solation for plugins
# (connectors, converters, transformations). The list should consist of top level dir
ectories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of
plugins and their dependencies
# Note: symlinks will be followed to discover dependencies or plugins.
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
#plugin.path=
```

- Example on how to modify the connector configuration file `connect-tablestore-sink-quickstart.properties`

The connector configuration file contains configuration items. These items include the connector class, Tablestore connection parameters, and data mapping. For more information, see [Configuration description](#).

```
# Specify the connector name.
name=tablestore-sink
# Specify the connector class.
connector.class=TableStoreSinkConnector
# Specify the maximum number of tasks.
tasks.max=1
# Specify the list of Kafka topics from which you want to export data.
topics=test
# Specify values for the following Tablestore connection parameters:
# The endpoint of the Tablestore instance.
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# The authentication mode.
tablestore.auth.mode=aksk
# The AccessKey ID and the AccessKey secret. If tablestore.auth.mode is set to aksk,
you need to specify the AccessKey ID and the AccessKey secret.
tablestore.access.key.id=xxx
tablestore.access.key.secret=xxx
# The name of the Tablestore instance.
tablestore.instance.name=xxx
## The configuration items related to Security Token Service (STS) authentication. If
STS authentication is used, the following configuration items must be specified. You
must also specify ACCESS_ID and ACCESS_KEY in environment variables when STS authenti
cation is used.
#sts.endpoint=
#region=
#account.id=
#role.name=
# Specify the format string for the name of the destination Tablestore table. You can
use <topic> in the string as a placeholder for the topic from which you want to expor
t data.
# topics.assign.tables is assigned higher priority than table.name.format. If topics.
assign.tables is specified, ignore the configuration of table.name.format.
# For example, if table.name.format is set to kafka_<topic> and the name of the Kafka
topic from which you want to export data is test, Kafka message records from the test
topic are mapped to the table named kafka_test in Tablestore.
table.name.format=<topic>
# Specify the mapping between the Kafka topic and the destination Tablestore table. T
he value must be in the <topic>:<tablename> format. The topic name and table name are
separated with a colon (:). If you want to specify multiple mappings, separate multip
le mappings with commas (,).
# If the mapping is not specified, the configuration of table.name.format is used.
# topics.assign.tables=test:test_kafka
# Specify whether to automatically create a destination table. Default value: false.
auto.create=true
# Specify how to process dirty data:
# An error may occur when the Kafka message records are parsed or written to the time
series table. You can specify the following two parameters to determine how to fix th
e error:
# Specify the fault tolerance capability. Valid values: none and all. Default value:
none.
```



```

# none: An error causes the data import task that uses Tablestore Sink Connector to fail.
# all: The message records for which errors are reported are skipped and logged.
runtime.error.tolerance=none
# Specify how dirty data is logged. Valid values: ignore, kafka, and tablestore. Default value: ignore.
# ignore: All errors are ignored.
# kafka: The message records for which errors are reported and the error messages are stored in a different Kafka topic.
# tablestore: The message records for which errors are reported and the error messages are stored in a Tablestore data table.
runtime.error.mode=ignore
# If you set runtime.error.mode to kafka, you must specify the Kafka cluster address and the topic.
# runtime.error.bootstrap.servers=localhost:9092
# runtime.error.topic.name=errors
# If you set runtime.error.mode to tablestore, you must specify the name of the Tablestore data table.
# runtime.error.table.name=errors
## The following configuration items are specific to data synchronization from Apache Kafka to time series tables in Tablestore.
# The connector working mode. Default value: normal.
tablestore.mode=timeseries
# Mappings of the primary key field in the time series table.
tablestore.timeseries.test.measurement=m
tablestore.timeseries.test.dataSource=d
tablestore.timeseries.test.tags=region,level
# Mappings of the time field in the time series table.
tablestore.timeseries.test.time=timestamp
tablestore.timeseries.test.time.unit=MILLISECONDS
# Specify whether to convert the column names of the time series data field to lowercase letters. Default value: true. The names of columns in the time series tables in the TimeSeries model do not support uppercase letters. If tablestore.timeseries.toLowerCase is set to false and the column name contains uppercase letters, an error is reported when data is written to the time series table.
tablestore.timeseries.toLowerCase=true
# Specify whether to store fields other than the primary key field and the time field as the time series data field in the time series table. Default value: true. If tablestore.timeseries.mapAll is set to false, only fields that are specified by using tablestore.timeseries.test.field.name are stored in the time series table as the time series data field.
tablestore.timeseries.mapAll=true
# Specify the name of the field that is contained in the time series data field. If you specify multiple fields that are contained in the time series data field, separate multiple field names with commas (,).
tablestore.timeseries.test.field.name=cpu
# Specify the type of the field that is contained in the time series data field. Valid values: double, integer, string, binary, and boolean.
# If multiple fields are contained in the time series data field, the field types and the field names must be configured in pairs. Separate multiple field types with commas (,).
tablestore.timeseries.test.field.type=double

```

2. Go to the \$KAFKA_HOME directory and run the following command to enable the standalone

mode:

```
bin/connect-standalone.sh config/connect-standalone.properties config/connect-tablestore-sink-quickstart.properties
```

Step 3: Generate message records


1. Go to the `$KAFKA_HOME` directory and run the following command to start a console producer client:

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

The following table describes the parameters that you need to configure to start a console producer client.

Parameter	Example	Description
<code>--broker-list</code>	localhost:9092	The address and port of the broker in the Kafka cluster.
<code>--topic</code>	test	The name of the topic. By default, a topic is automatically created when you start Tablestore Sink Connector. You can also manually create a topic.

2. Write messages to the topic named test.

 **Notice** To import data to a time series table, you must write data in the JSON format to the Kafka topic.

```
null
```

3. Log on to the [Tablestore console](#) to view the data.

1.1.4. Configuration description

Before you start Tablestore Sink Connector, you need to specify key-value pairs to pass parameters to the Kafka Connect process. This topic provides configuration examples and parameter descriptions to show how to configure Tablestore Sink Connector.

Configuration examples

The configuration items vary depending on whether data is synchronized from Kafka to a data table or a time series table in Tablestore. The configuration examples of the configuration files vary based on the working mode. This section provides an example on how to configure data synchronization from Kafka to a data table in Tablestore. To synchronize data to a time series table in Tablestore, you need to add configuration items that are specific to data synchronization from Kafka to a time series table in Tablestore.

- The following sample code provides an example on how to configure the configuration file in the .properties format for Tablestore Sink Connector in the standalone mode

```
# Specify the connector name.  
name=tablestore-sink
```

```

# Specify the connector class.
connector.class=TableStoreSinkConnector
# Specify the maximum number of tasks.
tasks.max=1
# Specify the list of Kafka topics from which data is exported.
topics=test
# Specify values for the following Tablestore connection parameters:
# The endpoint of the Tablestore instance.
tablestore.endpoint=https://xxx.xxx.ots.aliyuncs.com
# The AccessKey pair which consists of an AccessKey ID and an AccessKey secret.
tablestore.access.key.id=xxx
tablestore.access.key.secret=xxx
# The name of the Tablestore instance.
tablestore.instance.name=xxx
# Specify the following data mapping parameters:
# Specify the parser that is used to parse Kafka message records.
# The DefaultEventParser of Tablestore Sink Connector supports the Struct and Map classes
of Kafka Connect. You can also use a custom EventParser.
event.parse.class=com.aliyun.tablestore.kafka.connect.parsers.DefaultEventParser
# Specify the format string for the name of the destination Tablestore table. <topic> can
be used in the string as a placeholder for the topic from which you want to export data.
# topics.assign.tables is assigned a higher priority than table.name.format. If topics.as
sign.tables is specified, ignore the configuration of table.name.format.
# For example, if table.name.format is set to kafka_<topic> and the name of the Kafka top
ic from which you want to export data is test, Kafka message records from the test topic
are mapped to the table named kafka_test in Tablestore.
table.name.format=<topic>
# Specify the mapping between the Kafka topic and the destination Tablestore table. The v
alue must be in the <topic>:<tablename> format. The topic name and table name are separat
ed with a colon (:). If you want to specify multiple mappings, separate them with commas
(,).
# If the mapping is not specified, the configuration of table.name.format is used.
# topics.assign.tables=test:test_kafka
# Specify the primary key mode. Valid values: kafka, record_key, and record_value. Defaul
t value: kafka.
# kafka: <connect_topic>_<connect_partition> and <connect_offset> are used as the primary
key of the data table.
# record_key: Fields in the record keys are used as the primary key of the data table.
# record_value: Fields in the record values are used as the primary key of the data table
.
primarykey.mode=kafka
# Specify the name and data type of the primary key column in the destination Tablestore
data table.
# The format of the primary key column name is tablestore.<tablename>.primarykey.name. Th
e format of the data type of the primary key column is tablestore.<tablename>.primarykey.
type.
# <tablename> is a placeholder for the data table name.
# If the primary key mode is kafka, you do not need to specify the name and data type of
the primary key column. The default primary key column names {"topic_partition","offset"}
and the default data types {string, integer} of the primary key columns are used.
# If the primary key mode is record_key or record_value, you must specify the name and da
ta type of the primary key column.
# tablestore.test.primarykey.name=A,B
# tablestore.test.primarykey.type=string, integer
# Specify an attribute column whitelist to filter the fields in the record values to obt

```

```

# Specify an attribute column whitelist to filter the fields in the record values to obtain the required attribute columns.
# By default, the attribute column whitelist is empty. All fields in the record values are used as the attribute columns of the data table.
# The format of the attribute column name is tablestore.<tablename>.columns.whitelist.name. The format of the data type of the attribute column is tablestore.<tablename>.columns.whitelist.type.
# <tablename> is a placeholder for the data table name.
# tablestore.test.columns.whitelist.name=A,B
# tablestore.test.columns.whitelist.type=string,integer
# Specify how to write Kafka message records to the destination Tablestore table:
# Specify the write mode. Valid values: put and update. Default value: put.
# put: Data in the destination table is overwritten by Kafka message records.
# update: Data in the destination table is updated by Kafka message records.
insert.mode=put
# Specify whether to write data in the sequence that data is read. Default value: true. You can disable this option to improve the write performance.
insert.order.enable=true
# Specify whether to automatically create a destination table. Default value: false.
auto.create=false
# Specify the delete mode. Valid values: none, row, column, and row_and_column. Default value: none.
# none: No delete operations can be performed.
# row: Rows can be deleted.
# column: Attribute columns can be deleted.
# row_and_column: Rows and attribute columns can be deleted.
delete.mode=none
# Specify the maximum number of rows that can be included in the buffer queue in the memory when data is written to the data table. Default value: 1024. The value of this parameter must be an exponent of 2.
buffer.size=1024
# Specify the number of callback threads that are used when data is written to the data table. Default value = Number of vCPUs + 1.
# max.thread.count=
# Specify the maximum number of concurrent write requests that can be sent to write data to the data table. Default value: 10.
max.concurrency=10
# Specify the number of buckets to which data is written. Default value: 3. If you increase the value of this parameter, the concurrent write capability can be increased. However, you cannot set the value of this parameter to a value greater than the maximum number of concurrent write requests that you specified.
bucket.count=3
# Specify the interval at which the buffer queue is refreshed when data is written to the data table. Unit: milliseconds. Default value: 10000.
flush.Interval=10000
# Specify how to process dirty data:
# An error may occur when the Kafka message records are parsed or written to the data table. You can specify the following two parameters to determine how to fix the error:
# Specify the fault tolerance capability. Valid values: none and all. Default value: none.
# none: An error causes the data import task that uses Tablestore Sink Connector to fail.
# all: The message records for which errors are reported are skipped and logged.
runtime.error.tolerance=none
# Specify how dirty data is loaded. Valid values: ignore, kafka, and tablestore. Default

```

```

// Specify how data is logged. Valid values: ignore, kafka, and tablestore. Default
value: ignore.
# ignore: All errors are ignored.
# kafka: The message records for which errors are reported and the error messages are sto
red in a different Kafka topic.
# tablestore: The message records for which errors are reported and the error messages ar
e stored in a different Tablestore data table.
runtime.error.mode=ignore
# If you set runtime.error.mode to kafka, you must specify the Kafka cluster address and
the topic.
# runtime.error.bootstrap.servers=localhost:9092
# runtime.error.topic.name=errors
# If you set runtime.error.mode to tablestore, you must specify the name of the Tablestor
e data table.
# runtime.error.table.name=errors

```

- The following sample code provides an example on how to configure the configuration file in the .json format for Tablestore Sink Connector in the distributed mode:

```

{
  "name": "tablestore-sink",
  "config": {
    // Specify the connector class.
    "connector.class": "TableStoreSinkConnector",
    // Specify the maximum number of tasks.
    "tasks.max": "3",
    // Specify the list of Kafka topics from which you want to export data.
    "topics": "test",
    // Specify values for the following Tablestore connection parameters:
    // The endpoint of the Tablestore instance.
    "tablestore.endpoint": "https://xxx.xxx.ots.aliyuncs.com",
    // The AccessKey pair which consists of an AccessKey ID and an AccessKey secret.
    "tablestore.access.key.id": "xxx",
    "tablestore.access.key.secret": "xxx",
    // The name of the Tablestore instance.
    "tablestore.instance.name": "xxx",
    // Specify the following data mapping parameters:
    // Specify the parser that is used to parse Kafka message records.
    // The DefaultEventParser of Tablestore Sink Connector supports the Struct and Map cl
asses of Kafka Connect. You can also use a custom EventParser.
    "event.parse.class": "com.aliyun.tablestore.kafka.connect.parsers.DefaultEventParser",
    // Specify the format string for the name of the destination Tablestore table. <topic
> can be used in the string as a placeholder for the topic from which you want to export
data.
    // topics.assign.tables is assigned a higher priority than table.name.format. If topi
cs.assign.tables is specified, ignore the configuration of table.name.format.
    // For example, if table.name.format is set to kafka_<topic> and the name of the Kafk
a topic from which you want to export data is test, Kafka message records from the test t
opic are mapped to the table named kafka_test in Tablestore.
    "table.name.format": "<topic>",
    // Specify the mapping between the Kafka topic and the destination Tablestore table.
The value must be in the <topic>:<tablename> format. The topic name and table name are se
parated with a colon (:). If you want to specify multiple mappings, separate them with co
mmas (,).
    // If the mapping is not specified, the configuration of table.name.format is used.

```

```

// "topics.assign.tables":"test:test_kafka",
// Specify the primary key mode. Valid values: kafka, record_key, and record_value. Default value: kafka.
// kafka: <connect_topic>_<connect_partition> and <connect_offset> are used as the primary key of the data table.
// record_key: Fields in the record keys are used as the primary key of the data table.
// record_value: Fields in the record values are used as the primary key of the data table.
"primarykey.mode":"kafka",
// Specify the name and data type of the primary key column in the destination Tablestore data table.
// The format of the primary key column name is tablestore.<tablename>.primarykey.name. The format of the data type of the primary key column is tablestore.<tablename>.primarykey.type.
// <tablename> is a placeholder for the data table name.
// If the primary key mode is kafka, you do not need to specify the name and data type of the primary key column. The default primary key column names {"topic_partition","offset"} and the default data types {string, integer} of the primary key columns are used.
// If the primary key mode is record_key or record_value, you must specify the name and data type of the primary key column.
// "tablestore.test.primarykey.name":"A,B",
// "tablestore.test.primarykey.type":"string,integer",
// Specify an attribute column whitelist to filter the fields in the record values to obtain the required attribute columns.
// By default, the attribute column whitelist is empty. All fields in the record values are used as the attribute columns of the data table.
// The format of the attribute column name is tablestore.<tablename>.columns.whitelist.name. The format of the data type of the attribute column is tablestore.<tablename>.columns.whitelist.type.
// <tablename> is a placeholder for the data table name.
// "tablestore.test.columns.whitelist.name":"A,B",
// "tablestore.test.columns.whitelist.type":"string,integer",
// Specify how to write Kafka message records to the destination Tablestore table:
// Specify the write mode. Valid values: put and update. Default value: put.
// put: Data in the table is overwritten by Kafka message records.
// update: Data in the table is updated by Kafka message records.
"insert.mode":"put",
// Specify whether to write data in the sequence that data is read. Default value: true. You can disable this option to improve the write performance.
"insert.order.enable":"true",
// Specify whether to automatically create a destination table. Default value: false.

"auto.create":"false",
// Specify the delete mode. Valid values: none, row, column, and row_and_column. Default value: none.
// none: No delete operations can be performed.
// row: Rows can be deleted.
// column: Attribute columns can be deleted.
// row_and_column: Rows and attribute columns can be deleted.
"delete.mode":"none",
// Specify the maximum number of rows that can be included in the buffer queue in the memory when data is written to the data table. Default value: 1024. The value of this parameter must be an exponent of 2.
"max_row_in_buffer":"1024"

```

```

    "buffer.size":"1024",
    // Specify the number of callback threads that are used when data is written to the data table. Default value = Number of vCPUs + 1.
    // "max.thread.count":
    // Specify the maximum number of concurrent write requests that can be sent to write data to the data table. Default value: 10.
    "max.concurrency":"10",
    // Specify the number of buckets to which data is written. Default value: 3. You can increase the value of this parameter to increase the concurrent write capability. However, you cannot set the value of this parameter to a value greater than the maximum number of concurrent write requests that you specified.
    "bucket.count":"3",
    // Specify the interval at which the buffer queue is refreshed when data is written to the data table. Unit: milliseconds. Default value: 10000.
    "flush.interval":"10000",
    // Specify how to process dirty data:
    // An error may occur when the Kafka message records are parsed or written to the data table. You can specify the following two parameters to determine how to fix the error:
    // Specify the fault tolerance capability. Valid values: none and all. Default value: none.
    // none: An error causes the data import task that uses Tablestore Sink Connector to fail.
    // all: The message records for which errors are reported are skipped and logged.
    "runtime.error.tolerance":"none",
    // Specify how dirty data is logged. Valid values: ignore, kafka, and tablestore. Default value: ignore.
    // ignore: All errors are ignored.
    // kafka: The message records for which errors are reported and the error messages are stored in a different Kafka topic.
    // tablestore: The message records for which errors are reported and the error messages are stored in a different Tablestore data table.
    "runtime.error.mode":"ignore"
    // If you set runtime.error.mode to kafka, you must specify the Kafka cluster address and the topic.
    // "runtime.error.bootstrap.servers":"localhost:9092",
    // "runtime.error.topic.name":"errors",
    // If you set runtime.error.mode to tablestore, you must specify the name of the Tablestore data table.
    // "runtime.error.table.name":"errors",
  }

```

Parameters

The following table describes the parameters in the configuration file. You need to configure time series-related parameters only when you synchronize data from Kafka to a time series table in Tablestore.

Category	Parameter	Type	Required	Example	Description
	name	string	Yes	tablestore-sink	The name of the connector. The connector name must be unique.

Category	Parameter	Type	Required	Example	Description
Kafka Connect parameters	connector.class	class	Yes	TableStore SinkConnector	<p>The Java class of the connector.</p> <p>If you want to use the connector, specify the connector class by using connector.class. You can set connector.class to the full name or alias of the connector class. The full name of the connector class is com.aliyun.tablestore.kafka.connect.TableStoreSinkConnector and the alias of the connector class is TableStoreSinkConnector.</p> <pre>connector.class=com.aliyun.tablestore.kafka.connect.TableStoreSinkConnector</pre>
	tasks.max	integer	Yes	3	<p>The maximum number of tasks that can be created for the connector.</p> <p>If the maximum number of tasks fail to be created, fewer tasks may be created.</p>
	key.converter	string	No	org.apache.kafka.connect.json.JsonConverter	The key converter that is used to replace the default key converter that is specified in the worker configuration file.
	value.converter	string	No	org.apache.kafka.connect.json.JsonConverter	The value converter that is used to replace the default value converter that is specified in the worker configuration file.

Category	Parameter	Type	Required	Example	Description
	topics	list	Yes	test	<p>The list of Kafka topics that can be specified for the connector. Separate multiple Kafka topics with commas (,).</p> <p>You must specify topics to manage topics that are specified for the connector.</p>
Connector connection parameters	tablestore.endpoint	string	Yes	https://xxx.xxx.ots.aliyuncs.com	The endpoint of the Tablestore instance. For more information, see Endpoint .
	tablestore.mode	string	Yes	timeseries	<p>The type of the destination table. Default value: normal. Valid values:</p> <ul style="list-style-type: none">normal: a data table in Tablestore.timeseries: a time series table in Tablestore.
	tablestore.access.key.id	string	Yes	LTAn*****	The AccessKey ID and AccessKey secret of your account. For more information about how to obtain the AccessKey ID and the AccessKey secret, see Obtain an AccessKey pair .
	tablestore.access.key.secret	string	Yes	zbnK*****	

Category	Parameter	Type	Required	Example	Description
	tablestore.auth.mode	string	Yes	aksk	<p>The authentication mode. Default value: aksk. Valid values:</p> <ul style="list-style-type: none"> • aksk: uses the AccessKey ID and the AccessKey secret of an Alibaba Cloud account or a RAM user for authentication. In this topic, tablestore.auth.mode is set to aksk. • sts: uses the temporary access credentials that are obtained from Security Token Service (STS) for authentication. If Tablestore is connected to Message Queue for Apache Kafka, set tablestore.auth.mode to sts.
	tablestore.instance.name	string	Yes	myotstest	The name of the Tablestore instance.
					The Java class of the EventParser. Default value: DefaultEventParser. The parser parses Kafka message records to obtain the primary key column and attribute column of the data table.

Category	Parameter	Type	Required	Example	Description
	event.parse.class	class	Yes	DefaultEventParser	<p>Notice</p> <p>Tablestore provides limits on the size of column values. The values of primary key columns of the string or binary type cannot exceed 1 KB in size, and the values of attribute columns cannot exceed 2 MB in size. For more information, see General limits.</p> <p>If the column values exceed the limits after the data types are converted, the Kafka message records are processed as dirty data.</p> <p>To use DefaultEventParser, the keys or values of the Kafka message records must be of the Struct or Map class of Kafka Connect. The selected fields in Struct must be of data types that are supported by Tablestore Sink Connector. The fields are converted to data of the Tablestore data types based on the data type mapping table and then written to the data table. The data types of the values in Map must be the data types that are supported by Tablestore Sink Connector. Tablestore Sink Connector supports the same data types in Struct and Map. The values in Map are converted to data of the binary type and then written to the data table. For more information about the data type mappings between Kafka and Tablestore, see Appendix: Data type mappings between Kafka and Tablestore.</p>

Category	Parameter	Type	Required	Example	Description
	table.name.format	string	No	kafka_<topic>	<p>The format string for the name of the destination Tablestore data table. Default value: <topic>. <topic> can be used in the string as a placeholder for the topic from which you want to export data. For example, if table.name.format is set to kafka_<topic>, and the name of the Kafka topic from which you want to export data is test, the Kafka message records from the test topic are mapped to the table named kafka_test in Tablestore.</p> <p>topics.assign.tables is assigned a higher priority than table.name.format. If topics.assign.tables is specified, ignore the configuration of table.name.format.</p>
	topics.assign.tables	list	Yes	test:destTable	<p>Specifies the mapping between the topic and the destination Tablestore table in the <topic_1>: <tablename_1>, <topic_2>: <tablename_2> format. Separate multiple mappings with commas (,). For example, test:destTable specifies that the message records from the topic named test are written to the data table named destTable.</p> <p>topics.assign.tables is assigned a higher priority than table.name.format. If topics.assign.tables is specified, ignore the configuration of table.name.format.</p>

Category	Parameter	Type	Required	Example	Description
Data mapping parameters of the	primarykey.mode	string	No	kafka	<p>The primary key mode of the data table. Valid values:</p> <ul style="list-style-type: none"> • kafka: <connect_topic>_<connect_partition> and <connect_offset> are used as the primary key of the data table. The Kafka topic <connect_topic> and partition <connect_partition> are separated with an underscore (_), and <connect_offset> specifies the offset of the message record in the partition. • record_key: The fields of the Struct class or the keys of the Map class in the record keys are used as the primary key of the data table. • record_value: The fields of the Struct class or the keys of the Map class in the record values are used as the primary key of the data table. <p>Configure this parameter together with tablestore.<tablename>.primarykey.name and tablestore.<tablename>.primarykey.type. The value of this parameter is not case-sensitive.</p>
					<p>The primary key column name of the data table. <tablename> is a placeholder for the data table name. The value of this parameter contains one to four primary key column</p>

connector Category	Parameter	Type	Required	Example	Description names that are separated with commas (,).
	tablestore. <tablename >.primarykey.name	list	No	A,B	<p>The primary key column name varies with the primary key mode.</p> <ul style="list-style-type: none"> If the primary key mode is set to kafka, the default value of this parameter is <code>topic_partition,offset</code>. In the kafka primary key mode, you do not need to specify the primary key column names. Even if the primary key column names are specified, the default primary key column names take precedence. If the primary key mode is set to record_key, the fields of the Struct class or the keys of the Map class that have the same names as the specified primary key column names are extracted from the record keys as the primary key of the data table. In the record_key primary key mode, you must specify the primary key column names. If the primary key mode is set to record_value, the fields of the Struct class or the keys of the Map class that have the same names as the specified primary key column names are extracted from the record values as the primary key of the data table. In the record_value primary key mode, you must specify the primary key column names.

Category	Parameter	Type	Required	Example	Description
					<p>The primary key columns of the Tablestore data table are sequential. You need to take note of the sequence of primary key columns when you define tablestore.</p> <p><code><tablename>.primarykey.name</code> For example, PRIMARY KEY (A, B, C) and PRIMARY KEY (A, C, B) have different schemas.</p> <p>The data type of the primary key column in the data table <code><tablename></code> is a placeholder for the name of the data table. The value of this parameter contains one to four data types of the primary key columns. Separate data types of the primary key columns with commas (,). The sequence of data types of the primary key columns must correspond to the sequence of the primary key column names that are specified by tablestore.</p> <p><code><tablename>.primarykey.name</code>. The value of this parameter is not case-sensitive. Valid values: integer, string, binary, and auto_increment.</p> <p>The data type of the primary key column varies with the primary key mode.</p> <ul style="list-style-type: none"> If the primary key mode is set to kafka, the default value of this parameter is <code>string, integer</code>. If the primary key mode is set to record_key or record_value, you must specify the data types of the primary key columns.
	<code>tablestore.<tablename>.primarykey.type</code>	list	No	string, integer	<p>In the kafka primary key mode, you do not need to specify the data types of the primary key columns. Even if the data types of the primary key columns are specified, the default data types of the primary key columns take precedence.</p>

Category	Parameter	Type	Required	Example	Description
					<p>If the specified data type of the primary key column conflicts with the data type defined in the Kafka schema, a parse error occurs. In this case, you can configure Runtime Error parameters to fix the error.</p> <p>If this parameter is set to <code>auto_increment</code>, the field of the Kafka message records is inserted to the data table as an auto-increment primary key column when data is written to the data table.</p>
	<code>tablestore.<tablename>.columns.whitelist.name</code>	list	No	A,B	<p>The name of the attribute column in the attribute column whitelist. <code><tablename></code> is a placeholder for the data table name. Separate attribute column names with commas (,).</p> <p>If you do not configure this parameter, all fields of the Struct class or all keys of the Map class in the record values are used as the attribute columns of the data table. If you configure this parameter, the fields in the record values are filtered based on the specified attribute column whitelist to obtain the required attribute columns.</p>

Category	Parameter	Type	Required	Example	Description
	tablestore.<tablename>.columns.whitelist.type	list	No	string, integer	The data type of the attribute column in the attribute column whitelist. <tablename> is a placeholder for the data table name. Separate data types of the attribute columns with commas (,). The sequence of data types of the attribute columns must correspond to the sequence of the attribute column names that are specified by <tablename>.columns.whitelist.name. The value of this parameter is not case-sensitive. Valid values: integer, string, binary, boolean, and double.
	insert.mode	string	No	put	<p>The write mode. Default value: put. Valid values:</p> <ul style="list-style-type: none"> put: Existing data is overwritten by a row of data that you write to the table. This value corresponds to the PutRow operation of Tablestore. update: When you update a row of data, attribute columns are added to the row or the values of existing attribute columns are updated. This value corresponds to the UpdateRow operation of Tablestore. <p>The value of this parameter is not case-sensitive.</p>

Category	Parameter	Type	Required	Example	Description
	insert.order.enable	boolean	No	true	<p>Specifies whether data is written to the data table in the sequence that data is read. Default value: true. Valid values:</p> <ul style="list-style-type: none"> • true: Kafka message records are written to the data table in the sequence that message records are read. • false: Kafka message records are written to the data table without a specific sequence. This improves the write performance.
	auto.create	boolean	No	false	<p>Specifies whether to automatically create a destination table. A data table or a time series table can be automatically created. Default value: false. Valid values:</p> <ul style="list-style-type: none"> • true: The system automatically creates a destination Tablestore table. • false: The system does not automatically create a destination Tablestore table.
Connector write parameters					

Category	Parameter	Type	Required	Example	Description
	delete.mode	string	No	none	<p>The delete mode. The configuration of this parameter takes effect only when data is synchronized to a data table and the primary key mode is set to record_key. Default value: none. Valid values:</p> <ul style="list-style-type: none"> • none: No delete operations can be performed. • row: Rows can be deleted. If a record value is empty, the corresponding row is deleted. • column: Attribute columns can be deleted. If a field value of the Struct class or a key value of the Map class in the record values is empty, the corresponding attribute column is deleted. • row_and_column: Rows and attribute columns can be deleted. <p>The value of this parameter is not case-sensitive.</p> <p>This parameter is specified based on the value of the insert.mode parameter. For more information, see Appendix: Delete syntax.</p>
	buffer.size	integer	No	1024	<p>The maximum number of rows that can be included in the buffer queue in the memory when data is written to the data table. Default value: 1024. The value of this parameter must be an exponent of 2.</p>
	max.thread.count	integer	No	3	<p>The number of callback threads that are used when data is written to the data table. Default value = <code>Number of vCPUs + 1</code>.</p>

Category	Parameter	Type	Required	Example	Description
	max.concurrency	integer	No	10	The maximum number of concurrent write requests that can be sent to write data to the data table.
	bucket.count	integer	No	3	The number of buckets to which data is written. Default value: 3. If you increase the value of this parameter, the concurrent write capability can be increased. However, you cannot set the value of this parameter to a value greater than the maximum number of concurrent write requests that you specified.
	flush.Interval	integer	No	10000	The interval at which the buffer queue is refreshed when data is written to the data table. Unit: milliseconds. Default value: 10000.
	runtime.error.tolerance	string	No	none	<p>The error handling policy that is used if an error occurs when the Kafka message records are parsed or written to the table. Default value: none. Valid values:</p> <ul style="list-style-type: none"> • none: An error causes the data import task that uses Tablestore Sink Connector to fail. • all: The message records for which errors are reported are skipped and logged. <p>The value of this parameter is not case-sensitive.</p>
					<p>Specifies how to process the message records for which errors are reported when Kafka message records are parsed or written to the table. Default value: ignore. Valid values:</p> <ul style="list-style-type: none"> • ignore: All errors are

Category	Parameter	Type	Required	Example	ignored. Description
Connector Runtime Error parameters	runtime.error.mode	string	No	ignore	<ul style="list-style-type: none"> • kafka: The message records for which errors are reported and the error messages are stored in a different Kafka topic. In this case, you need to specify runtime.error.bootstrap.servers and runtime.error.topic.name. The keys and values of the Kafka message records for which errors are reported in the new topic are the same as those of the message records in the topic from which you want to export data. The ErrorInfo field is included in the header to log the error messages. • tablestore: The message records for which errors are reported and the error messages are stored in a different Tablestore data table. In this case, you need to specify runtime.error.table.name. The primary key columns of the data table that is used to log message records for which errors are reported and the error messages are topic_partition (string type) and offset (integer type). The attribute columns of the data table are key (bytes type), value (bytes type), and error_info (string type).

Category	Parameter	Type	Required	Example	Description
					If runtime.error.mode is set to kafka, you need to serialize the headers, keys, and values of the Kafka message records. If runtime.error.mode is set to tablestore, you need to serialize the keys and values of the Kafka message records. By default, org.apache.kafka.connect.json.JsonConverter is used to serialize data and schemas.enable is set to true. You can use JsonConverter to deserialize data to obtain the original data. For more information about Converter, see Kafka Converter .
	runtime.error.bootstrap.servers	string	No	localhost:9092	The address of the Kafka cluster where the message records for which errors are reported and the error messages are stored.
	runtime.error.topic.name	string	No	errors	The name of the Kafka topic that stores the message records for which errors are reported and the error messages.
	runtime.error.table.name	string	No	errors	The name of the Tablestore table that stores the message records for which errors are reported and the error messages.

Category	Parameter	Type	Required	Example	Description
	tablestore.timeseries.<tablename>.measurement	string	Yes	mName	<p>Specifies that the values that correspond to the specified key in JSON formatted data are written to the time series table as the values of the <code>_m_name</code> field.</p> <p>If <code>tablestore.timeseries.<tablename>.measurement</code> is set to <code><topic></code>, the values that correspond to the topic key of Kafka message records are written to the time series table as the values of the <code>_m_name</code> field.</p> <p><code><tablename></code> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements. For example, if the name of the time series table is <code>test</code>, the parameter name is <code>tablestore.timeseries.test.measurement</code>.</p>
	tablestore.timeseries.<tablename>.dataSource	string	Yes	ds	<p>Specifies that the values that correspond to the <code>ds</code> key in JSON formatted data are written to the time series table as the values of the <code>_data_source</code> field.</p> <p><code><tablename></code> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements.</p>

Category	Parameter	Type	Required	Example	Description
Time series-related parameters	tablestore.timeseries.<tablename>.tags	list	Yes	region,level	Specifies that the values that correspond to the region and level keys in JSON formatted data are written to the time series table as the values of the tags field. <tablename> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements.
	tablestore.timeseries.<tablename>.time	string	Yes	timestamp	Specifies that the values that correspond to the timestamp key in JSON formatted data are written to the time series table as the values of the _time field. <tablename> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements.
	tablestore.timeseries.<tablename>.time.unit	string	Yes	MILLISECONDS	The unit of the values of the tablestore.timeseries.<tablename>.time parameter. Valid values: SECONDS, MILLISECONDS, MICROSECONDS, and NANOSECONDS. <tablename> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements.

Category	Parameter	Type	Required	Example	Description
	tablestore.timeseries. <tablename>.field.name	list	No	cpu,io	Specifies that the cpu and io keys in JSON formatted data are written to the time series table as the names of _field_name and the values that correspond to the cpu and io keys in JSON formatted data are written to the time series table as the values of _field_name. <tablename> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements.
	tablestore.timeseries. <tablename>.field.type	string	No	double,integer	The data type of the field that is specified by tablestore.timeseries.<tablename>.field.name. Valid values: double, integer, string, binary, and boolean. Separate multiple data types with commas (.). <tablename> in the parameter is a placeholder for the name of the time series table. Modify the parameter name based on your business requirements.
	tablestore.timeseries.mapAll	boolean	No	false	Specifies whether fields other than the primary key fields and time fields in JSON formatted data are written to the time series table as fields. If tablestore.timeseries.mapAll is set to false, you must configure the tablestore.timeseries.<tablename>.field.name and tablestore.timeseries.<tablename>.field.type parameters.


Category	Parameter	Type	Required	Example	Description
	tablestore.timeseries.toLowerCase	boolean	No	true	Specifies whether the keys in the fields are converted to lowercase letters before being written/and then written to the time series table. The keys in the fields are keys in the non-primary key fields or non-time fields, or keys specified in tablestore.timeseries. <tablename>.field.name.
	tablestore.timeseries.rowsPerBatch	integer	No	50	The maximum number of rows that can be written to Tablestore in a request. The maximum and default values are 200.

Appendix: Data type mappings between Kafka and Tablestore

The following table describes the mappings between the data types of Kafka and Tablestore.

Kafka schema type	Tablestore data type
STRING	STRING
INT8, INT16, INT32, and INT64	INTEGER
FLOAT32 and FLOAT64	DOUBLE
BOOLEAN	BOOLEAN
BYTES	BINARY

Appendix: Delete syntax

 **Note** This feature is supported only when data is synchronized from Kafka to a data table in Tablestore.

The following table describes the methods that are used to write data to a Tablestore data table based on the configurations of the write mode (insert.mode) and delete mode (delete.mode) when message records contain empty values and data is synchronized from Kafka to a data table in Tablestore.

insert.mode	put				update			
delete.mode	none	row	column	row_and_column	none	row	column	row_and_column

insert.m ode	put				update			
Empty values	Overwrit e	Delete rows	Overwrit e	Delete rows	Dirty data	Delete rows	Dirty data	Delete rows
All empty fields in values	Overwrit e	Overwrit e	Overwrit e	Overwrit e	Dirty data	Dirty data	Delete columns	Delete columns
Some empty fields in values	Overwrit e	Overwrit e	Overwrit e	Overwrit e	Ignore empty values	Ignore empty values	Delete columns	Delete columns

1.1.5. Error handling

In some cases, when the Kafka data is imported to a Tablestore table, an error occurs. If you do not want a data import task that uses Tablestore Sink Connector to immediately fail when an error occurs, you can configure an error handling policy.

The following errors may occur:

- **Kafka Connect Error**

This error occurs before the data import task that uses Tablestore Sink Connector is executed. For example, this error can occur when you use Converter for deserialization or use [Kafka Transformations](#) to perform lightweight modification on message records. If this error occurs, you can configure the error handling option that is provided by Kafka.

If you want to skip this error, specify `errors.tolerance=all` in the connector configuration file. For more information, see [Kafka Connect Configs](#).

- **Tablestore Sink Task Error**

This error occurs when the data import task that uses Tablestore Sink Connector is in progress. For example, this error can occur when message records are being parsed or message records are being written to Tablestore. If this error occurs, you can configure the error handling option that is provided by Tablestore Sink Connector.

If you want to skip this error, specify `errors.tolerance=all` in the connector configuration file. For more information, see [Configuration description](#). You can also specify the method that is used to report errors. If you want to save the message records for which errors are reported and the error messages to a different data table in Tablestore, configure the following parameters:

```
runtime.error.tolerance=all
runtime.error.mode=tablestore
runtime.error.table.name=error
```

In the distributed mode, you can use the REST API to manage the connector and task. If the connector or task stops because an error occurs, you can manually restart the connector or task.

- i. Check the connector and task status.

- Check the connector status:

```
curl http://localhost:8083/connectors/{name}/status
```

- Check the task status:

```
curl http://localhost:8083/connectors/{name}/tasks/{taskid}/status
```

In the preceding commands, `http://localhost:8083/connectors` is the address of the Kafka REST service, the value of `name` must be the same as the connector name in the configuration file, and `taskid` is included in the connector status information.

You can run the following command to obtain the `taskid` value:

```
curl http://localhost:8083/connectors/{name}/tasks
```

- ii. Manually restart the connector or the task.

- Restart the connector:

```
curl -X POST http://localhost:8083/connectors/{name}/restart
```

- Restart the task:


```
curl -X POST http://localhost:8083/connectors/{name}/tasks/{taskId}/restart
```

1.2. Synchronize data from one table to another table

This topic describes how to synchronize data from one table to another table by using Tunnel Service, DataWorks, or DataX.

Prerequisites

The destination table is created. For more information, see [Create tables](#).

 **Notice** The destination table must contain the columns you want to synchronize from the source table.

Use Tunnel Service to synchronize data

After the tunnel of the source table is created, you can use Tablestore SDK to synchronize data from the source table to the destination table. You can customize logic to process data for the business during synchronization.

1. Create the tunnel of the source table in the Tablestore console or by using Tablestore SDK. Record the tunnel ID. For more information, see [Quick start](#) or [SDK usage](#).
2. Synchronize data by using Tablestore SDK.

The following code provides an example on how to synchronize data by using Tablestore SDK:

```

public class TunnelTest {
    public static void main(String[] args){
        TunnelClient tunnelClient = new TunnelClient("endpoint",
            "accessKeyId","accessKeySecret","instanceName");
        TunnelWorkerConfig config = new TunnelWorkerConfig(new SimpleProcessor());
        // You can view the tunnel ID on the Tunnels tab of the console or query the tu
        nnel ID by calling describeTunnelRequest.
        TunnelWorker worker = new TunnelWorker("tunnelId", tunnelClient, config);
        try {
            worker.connectAndWorking();
        } catch (Exception e) {
            e.printStackTrace();
            worker.shutdown();
            tunnelClient.shutdown();
        }
    }
    public static class SimpleProcessor implements IChannelProcessor{
        // Connect the tunnel to the destination table.
        TunnelClient tunnelClient = new TunnelClient("endpoint",
            "accessKeyId","accessKeySecret","instanceName");
        @Override
        public void process(ProcessRecordsInput processRecordsInput) {
            // Incremental or full data is returned in ProcessRecordsInput.
            List<StreamRecord> list = processRecordsInput.getRecords();
            for(StreamRecord streamRecord : list){
                switch (streamRecord.getRecordType()){
                    case PUT:
                        // Customize the logic to process data for the business.
                        //putRow
                        break;
                    case UPDATE:
                        //updateRow
                        break;
                    case DELETE:
                        //deleteRow
                        break;
                }
                System.out.println(streamRecord.toString());
            }
        }
        @Override
        public void shutdown() {
        }
    }
}

```


Use DataWorks or DataX to synchronize data

You can use DataWorks or DataX to synchronize data from the source table to the destination table. This section describes how to synchronize data by using DataWorks.

1. Add data sources of Tablestore

Add the Tablestore instances of the source table and the destination table as data sources. For more information, see [Add a data source](#).


2. Create a synchronization task node.
 - i. Log on to the [DataWorks console](#) as the project administrator.


 **Note** Only the project administrator role can add data sources. Members who assume other roles can only view data sources.



- ii. Select a region. In the left-side navigation pane, click **Workspaces**.
 - iii. On the **Workspaces** page, find the workspace and click **Data Analytics** in the Actions column.
 - iv. On the **Data Analytics** page of the DataStudio console, click **Business Flow** and select a business flow.

For more information about how to create a business flow, see [Create a workflow](#).

- v. Right-click **Data Integration** and choose **Create > Batch synchronization**.
 - vi. In the **Create Node** dialog box, enter a node name in the **Node Name** field.
3. Configure the data source.
 - i. Click **Data Integration**. Double-click the name of the node for the data synchronization task.
 - ii. On the editing page for the node of data synchronization, find the **Connections** section and set the Source connection and the Target connection.

Set **Connection type** to **OTS** for both the **Source** and **Target** connections. Then, select connections for Source and Target. Click the  icon or the **Switch to Code Editor** icon to configure a script.

 **Note** Tablestore supports only the script mode. For more information about how to configure scripts, see [Configure Tablestore Reader](#) and [Configure Tablestore Writer](#).

- iii. Click the  icon to save the data source configurations.
4. Run the synchronization task.
 - i. Click the  icon.
 - ii. In the **Arguments** dialog box, select the resource group for scheduling.
 - iii. Click **OK** to run the task.

After the task is completed, you can check whether the task is successful and view the number of exported rows on the **Runtime Log** tab.

5. (Optional) Execute the synchronization task at the scheduled time. For more information, see [Configure recurrence and dependencies for a node](#).

2.Data export

2.1. Synchronize data from tablestore to OSS

2.1.1. Overview

You can use scripts in the Data Integration console to synchronize incremental and full data from Tablestore to OSS.

Tablestore is a distributed NoSQL database service that allows you to store data based on the Apsara distributed system of Alibaba Cloud. Tablestore is designed to provide 99.99% availability and 99.999999999% (eleven 9's) data reliability. Tablestore adopts sharding and load balancing technologies to scale out services and handle concurrent transactions. You can use Tablestore to store and query large amounts of structured data in real time.

Object Storage Service (OSS) is a massive-volume, secure, low-cost, and highly-reliable cloud storage service. It provides 99.99999999% data reliability. You can use RESTful API for storage and access in any place on the Internet. Its capacity and processing capability can be elastically scaled, and multiple storage modes are provided, comprehensively optimizing the storage cost.

Scenarios

Tablestore: Provides professional data-persistent storage service and user-oriented real-time read/write operations with high concurrency and low latency.

OSS: Supports backup at an extremely low cost.

Usage

- **Write**
Data can be directly written to Tablestore.
- **Read**
Data can be directly read from Tablestore.
- **Backup**
Automatic backup is supported.
- **Restoration**
Data can be re-written to Tablestore by using Data Integration (OSSReader and OTSWriter).

Limits

- **Write by whole rows**
Tablestore Stream requires that a whole row of data be written to Tablestore each time. Currently, the whole-row data write mode is applied to the writing of time sequence data such as IoT data. Therefore, data cannot be modified subsequently.
- **Synchronization latency**

Currently, periodic scheduling is used and the scheduling interval is 5 minutes. The plugin has a latency of 5 minutes and the total latency of a synchronization task is 5 to 10 minutes.

Activation

- [Activate Tablestore](#)
- [Activate OSS](#)

Data tunnel

Offline

- Export the full data to OSS.
 - [Script mode](#)
- Synchronize data to OSS in incremental mode.
 - [Script mode](#)
- Fully import data into Tablestore.
 - [Script mode](#)


2.1.2. Export full data in script mode

This topic describes how to use the DataWorks console to synchronize full data from Tablestore to Object Storage Service (OSS). The objects synchronized to OSS can be downloaded and stored in OSS as the backup of the data in Tablestore.

Step 1: Add a Tablestore data source

To add a Tablestore data source, perform the following steps:

1. Go to the Data Integration homepage.
 - i. Log on to the [DataWorks console](#) as a project administrator.

 **Note** Only the project administrator role can be used to add data sources. Members who assume other roles can only view data sources.

- ii. Select a region. In the left-side navigation pane, click **Workspaces**.
 - iii. On the **Workspaces** page, click **Data Integration** in the Actions column that corresponds to the required workspace.
2. Add a data source.
 - i. In the Data Integration console, choose **Data Source** > Data Sources.
 - ii. On the **Data Source** page, click **Add data source** in the upper-right corner.
 - iii. In the **Add data source** dialog box, click **OTS** in the **NoSQL** section.
 - iv. In the **Add OTS data source** dialog box, configure the parameters.

Add OTS data source

* Data Source Name :

Custom name

Data source description :

Network connection :

Please Select

type

* Endpoint :

?

* Table Store instance :

name

* AccessKey ID :

?

* AccessKey Secret :

Resource Group :

Data Integration

Schedule

?

If your Data Integration task used this connector, it is necessary to ensure that the connector can be connected by the corresponding resource group. Please refer to the [resource group](#) for detailed concepts and [network solutions](#).

View current best network solution recommendations

Resource group name	Type	Connectivity status (Click status to view details)	Test time	Operation
---------------------	------	---	-----------	-----------

Previous step

Complete

Parameter	Description
Data source	The name of the data source.
Description	The description of the data source.
Endpoint	<p>The endpoint of the Tablestore instance. For more information, see Endpoint.</p> <ul style="list-style-type: none"> ■ If the Tablestore instance is in the same region as the OSS bucket, enter the endpoint to access the Tablestore instance over the classic network. ■ If the Tablestore instance and the OSS bucket are located in different regions, enter the public IP address. ■ Do not enter a virtual private cloud (VPC) endpoint.
Table Store instance name	The name of the Tablestore instance.
AccessKey ID	The AccessKey ID and AccessKey secret of your logon account. For more information about how to obtain the AccessKey ID and AccessKey secret, see Create an AccessKey pair for a RAM user .
AccessKey Secret	

- v. Click **Test connectivity** to test the connectivity of the data source.
3. Click **Complete**.

On the **Data Source** page, information about the data source appears.

Step 2: Add an OSS data source

The operations are similar to those of Step 1. However, in this step, you must click **OSS** in the **Semi-structured storage** section.

Note When you configure an OSS data source, make sure that the endpoint does not contain the bucket name.

In this example, the following figure shows that the data source is named OTS2OSS.

Add OSS data source

* Data Source Name :

OTS2OSS

Data source description :

Network connection :

public

type

* Endpoint :

https://oss-cn-hangzhou.aliyuncs.com

?

* Bucket :

myhotstest

?

* AccessKey ID :

?

* AccessKey Secret :

Resource Group :

Data Integration

Schedule

?

If your Data Integration task used this connector, it is necessary to ensure that the connector can be connected by the corresponding resource group. Please refer to the [resource group](#) for detailed concepts and [network solutions](#).

View current best network solution recommendations

Resource group name	Type	Connectivity status (Click status to view details)	Test time	Operation
Public Resource Group		Connectable	Dec 31, 2020 15:09:56	Test connectivity

Precautions

The connectivity testing may fail due to the following possible causes. Troubleshoot the failures as instructed.

1

The data source is not started. Make sure that the data source is started

Previous step

Complete

Step 3: Create a synchronization task

To create and configure a task to synchronize data from Tablestore to OSS, perform the following steps:

1. Go to Data Analytics.
 - i. Log on to the [DataWorks console](#) as a project administrator.
 - ii. Select a region. In the left-side navigation pane, click **Workspaces**.


- iii. On the **Workspaces** page, click **Data Analytics** in the Actions column that corresponds to the workspace.
2. On the **Data Analytics** page of the DataStudio console, click **Business Flow** and select a business flow.

For more information about how to create a business flow, see [Create a workflow](#).

3. Create a synchronization task node.

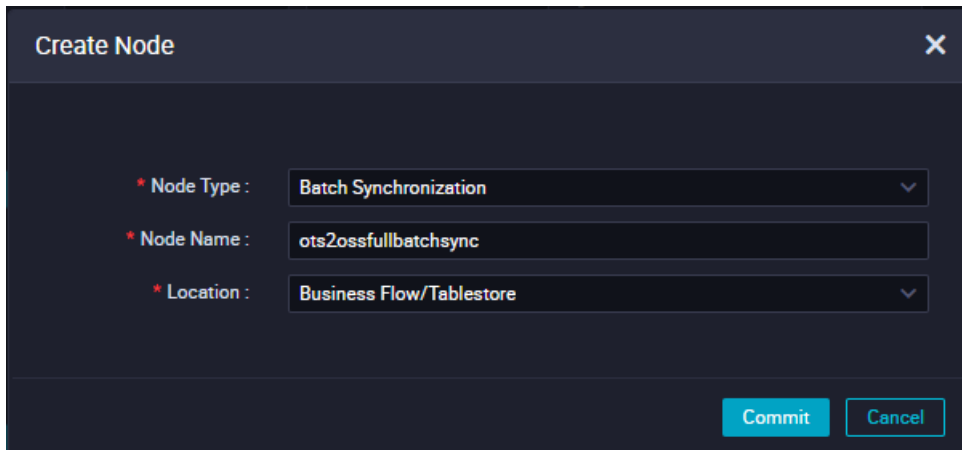
You must create a node for each synchronization task.

- i. Right-click **Data Integration** and then choose **Create > Batch synchronization**.

You can also move the pointer over the  icon, and then choose **Data Integration >**

Batch synchronization to create a node.

- ii. In the **Create Node** dialog box, configure Node Name and Location.



- iii. Click **Commit**.
4. Configure the Tablestore data source.
 - i. Click **Data Integration**. Double-click the name of the node for the data synchronization task.

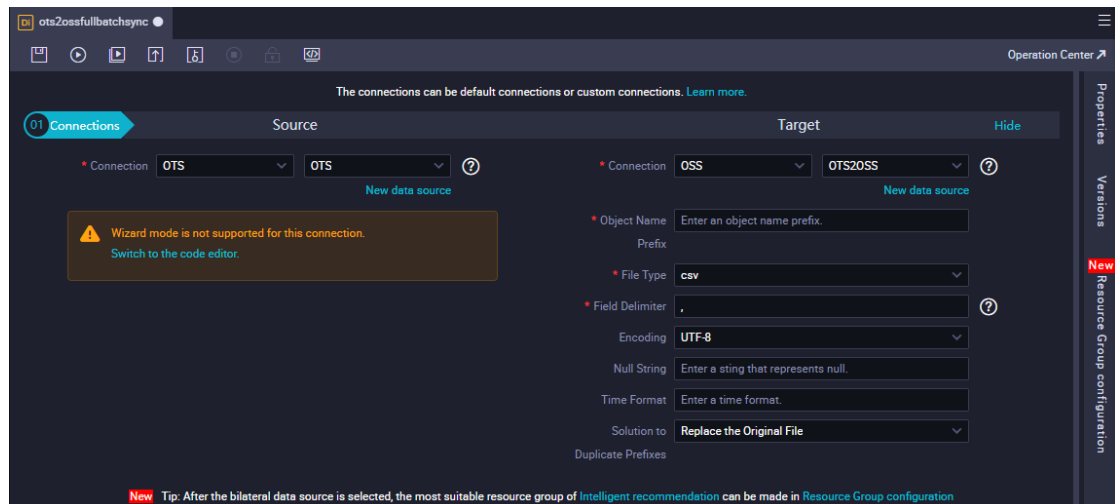
- ii. On the edit page of the synchronization task node, configure Source and Target in the **Connections** section.


- Configure Source.

Set **Connection** to **OTS** for **Source**.

- Configure Target.

Set **Connection** to **OSS** for **Target**. Configure the data source.



- iii. Click the  icon or **Switch to the code editor** to configure the script.

Tablestore supports only the script mode to configure the connection. When you use the script to configure the connection, you must configure Tablestore Reader and OSS Writer plug-ins. For more information about specific operations, see [Tablestore Reader](#) and [OSS Writer](#).

On the configuration page of the script, configure the parameters based on the following example:

```
{
  "type": "job",    # Do not change the value.
  "version": "1.0", # Do not change the value.
  "configuration": {
    "setting": {
      "errorLimit": {
        "record": "0" # When the number of errors exceeds the value of record, the task fails to be imported.
      },
      "speed": {
        "mbps": "1", # The rate at which to import data. Unit: MB/s.
        "concurrent": "1" # The number of concurrent threads.
      }
    },
    "reader": {
      "plugin": "ots", # Do not change the value.
      "parameter": {
        "datasource": "", # The name of the data source from which data is integrated. You must configure the name of the data source before data is integrated. You can select a data source of Tablestore or enter authentication information such as the AccessKey ID in plaintext. We recommend that you use a data source.
      }
    }
  }
}
```

```


    "table": "", # The name of the data table in Tablestore.
    "column": [ # Required. The names of the columns you want to export to OSS.
        {
            "name": "column1" # The name of the column in Tablestore. This column i
s to be exported to OSS.
        },
        {
            "name": "column2" # The name of the column in Tablestore. This column is
to be exported to OSS.
        }
    ],
    "range": {
        "begin": [
            {
                "type": "INF_MIN" # The starting position of the first primary key col
umn in Tablestore. To export full data, set the parameter to INF_MIN. To export onl
y part of the data, set the parameter based on your requirements. If the table cont
ains multiple primary key columns, configure the information about the correspondin
g primary key columns for the begin parameter.
            }
        ],
        "end": [
            {
                "type": "type": "INF_MAX" # The ending position of the first primary k
ey column in Tablestore. To export full data, set the parameter to INF_MAX. To expo
rt part of data, set the parameter based on your requirements. If the data table co
ntains multiple primary key columns, configure the information about the correspond
ing primary key columns for the end parameter.
            }
        ],
        "split": [ # Configure the information about the partitions of the Tablesto
re data table. You can use this feature to accelerate data export. This parameter i
s automatically configured in the next version.
        ]
    }
},
"writer": {
    "plugin": "oss",
    "parameter": {
        "datasource": "", # Configure the OSS data source.
        "object": "", # The prefix of the object name. The prefix excludes bucket nam
es. Example: tablestore/20171111/. To perform scheduled export, you must use variab
les in the prefix. Example of a variable: tablestore/${date}. Then, specify the ${d
ate} value when you configure scheduling parameters.
        "writeMode": "truncate", # The operation the system performs when files of the
same name exist. Use truncate to export full data. Valid values: truncate, append,
and nonConflict. truncate specifies that files of the same name are cleared. append
specifies that data is appended to the content of files of the same name. nonConfli
ct specifies that an error is reported if files of the same name exist.
        "fileFormat": "csv", # The format of the file. Valid values: csv, txt, and par
quet.
        "encoding": "UTF-8", # The encoding type.
        "nullFormat": "null", # The string used to define the null value. The value ca

```

```

n be an empty string.
    "dateFormat": "yyyy-MM-dd HH:mm:ss", # The time format.
    "fieldDelimiter": ",", # The delimiter used to separate each column.
  }
}
}
}


```

iv. Click the  icon to save the data source configurations.

Note

- Full data export is used to export all data at a time. Therefore, you do not need to configure scheduling parameters. To configure scheduling parameters, configure scheduling parameters in [Synchronize incremental data](#).
- If the script configurations contain variables such as `${date}`, set the variable to a specific value when you run the task to synchronize data.

5. Run the synchronization task.

- Click the  icon.
- In the **Arguments** dialog box, select the resource group for scheduling.
- Click **OK** to run the task.

After the task is run, you can check whether the task was successful and the number of rows of exported data on the **Runtime Log** tab.

Step 4: View the data exported to OSS

- Log on to the [OSS console](#).
- Select the corresponding bucket and object name. You can check whether the object contains the content as expected after you download the object.

2.1.3. Synchronize incremental data in script mode

This topic describes how to use the DataWorks console to synchronize incremental data from Tablestore to Object Storage Service (OSS).

Step 1: Add a Tablestore data source

If a Tablestore data source is added, skip this step.

For more information about how to add Tablestore data sources, see [Step 1: Add a Tablestore data source](#).

Step 2: Add an OSS data source


If an OSS data source is added, skip this step.

For more information about how to add an OSS data source, see [Step 2: Add an OSS data source](#).

Step 3: Configure a scheduled synchronization task

To create and configure a task to synchronize incremental data from Tablestore to OSS, perform the following steps.

1. Go to Data Analytics.
 - i. Log on to the [DataWorks console](#) as a project administrator.

 **Note** Only the project administrator role can be used to add data sources. Members who assume other roles can only view data sources.

- ii. Select a region. In the left-side navigation pane, click **Workspaces**.
 - iii. On the **Workspaces** page, click **Data Analytics** in the Actions column that corresponds to the workspace.
2. On the **Data Analytics** page of the DataStudio console, click **Business Flow** and select a business flow.

For more information about how to create a business flow, see [Create a workflow](#).

3. Create a synchronization task node.

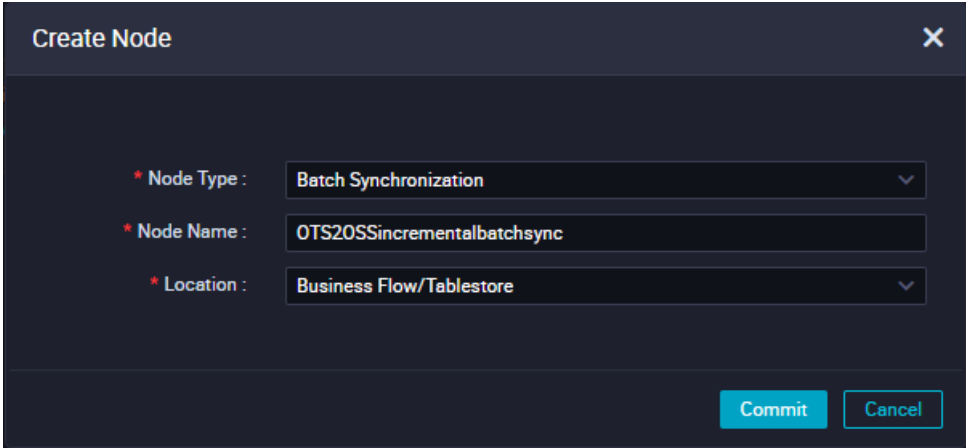
You must create a node for each synchronization task.

- i. Right-click **Data Integration** and then choose **Create > Batch synchronization**.

You can also move the pointer over the  icon, and then choose **Data Integration >**

Batch synchronization to create a node.

- ii. In the **Create Node** dialog box, configure Node Name and Location.



The image shows a 'Create Node' dialog box with a dark background. It has a title bar with 'Create Node' and a close button. Inside, there are three fields: 'Node Type' with a dropdown menu showing 'Batch Synchronization', 'Node Name' with a text input field containing 'OTS2OSSincrementalbatchsync', and 'Location' with a dropdown menu showing 'Business Flow/Tablestore'. At the bottom right, there are two buttons: 'Commit' and 'Cancel'.

- iii. Click **Commit**.
4. Configure the Tablestore data source.
 - i. Click **Data Integration**. Double-click the name of the node for the data synchronization task.


- ii. On the edit page of the synchronization task node, configure Source and Target in the **Connections** section.

- Configure Source.

Set **Connection** to **OTS Stream** for **Source**. Select a data source and table. Configure the start time and end time of the task, the name of the status table, and the maximum number of retry attempts.

- Configure Target.

Set **Connection** to **OSS** for **Target**. Select a data source. Configure the prefix of the object name, text type, and delimiter for the column.

- iii. Click the  icon to configure the script.

When you use the script to configure the connection, you must configure OTSStream Reader and OSS Writer plug-ins. For more information about specific operations, see [Tablestore Reader](#) and [OSS Writer](#).

On the configuration page of the script, configure the parameters based on the following example:

```
{
  "type": "job",
  "version": "1.0",
  "configuration": {
    "setting": {
      "errorLimit": {
        "record": "0" # The maximum number of errors that are allowed. The synchronization task fails when the number of errors exceeds this value.
      },
      "speed": {
        "mbps": "1", # The maximum bandwidth of each synchronization task.
        "concurrent": "1" # The maximum number of concurrent threads for each synchronization task.
      }
    },
    "reader": {
      "plugin": "otsstream", # The name of the Reader plug-in.
    }
  }
}
```



```


"parameter": {
  "datasource": "", # The name of the Tablestore data source. If you specify datasource, you can leave the endpoint, accessId, accessKey, and instanceName parameters empty.
  "dataTable": "", # The name of the data table in Tablestore.
  "statusTable": "TablestoreStreamReaderStatusTable", # The table that stores the status of Tablestore Stream. In most cases, you do not need to change the value of this parameter.
  "startTimestampMillis": "", # The start time of data export. The task must be started in loops because this task is used for incremental export. The start time for each loop is different. Therefore, you must set a variable such as ${start_time}.
  "endTimestampMillis": "", # The time at which data export ends. You must set this parameter to a variable such as ${end_time}.
  "date": "yyyyMMdd", # The date based on which to export data. The same results are returned if you configure the date parameter while the startTimestampMillis and endTimestampMillis parameters are also configured. If you configure startTimestampMillis and endTimestampMillis, you can delete the date parameter.
  "mode": "single_version_and_update_only", # The mode in which Tablestore Stream is used to export data. Set this parameter to single_version_and_update_only. If the configuration template does not contain this parameter, add this parameter.
  "column": [ # Set the columns that you want to export from the data table to OSS. If the configuration template does not contain this parameter, add this parameter. You can customize the number of columns.
    {
      "name": "uid" # The name of a primary key column in the data table of Tablestore.
    },
    {
      "name": "name" # The name of an attribute column in the data table of Tablestore.
    }
  ],
  "isExportSequenceInfo": false, # Specify whether to export time series information. If you set the mode parameter to single_version_and_update_only, this parameter can be set only to false.
  "maxRetries": 30 # The maximum number of retry attempts.
},
"writer": {
  "plugin": "oss", # The name of the Writer plug-in.
  "parameter": {
    "datasource": "", # The name of the OSS data source.
    "object": "", # The prefix of the name of the file you want to synchronize to OSS. We recommend that you use the "Tablestore instance name/Table name/date" format. Example: "instance/table/{date}".
    "writeMode": "truncate", # The operation the system performs when files of the same name exist. Valid values: truncate, append, and nonConflict. truncate specifies that files of the same name are deleted. append specifies that data is appended to the content of files of the same name. nonConflict specifies that an error is reported if files of the same name exist.
    "fileFormat": "csv", # The format of the file. Valid values: csv, txt, and parquet.
    "encoding": "UTF-8", # The encoding type.
    "nullFormat": "null", # The string used to define the null value. The value can be an empty string.
  }
}

```

```

an empty string.
    "dateFormat": "yyyy-MM-dd HH:mm:ss", # # The time format.
    "fieldDelimiter": ",", # The delimiter used to separate each column.
  }
}
}
}

```

iv. Click the  icon to save the data source configurations.

5. Run the synchronization task.

i. Click the  icon.

ii. In the **Arguments** dialog box, select the resource group for scheduling.

iii. Click **OK** to run the task.

After the task is completed, you can check whether the task is successful and view the number of exported rows on the **Runtime Log** tab.

Incremental data is automatically synchronized from Tablestore to OSS at the latency of 5 to 10 minutes.

6. Configure the scheduling parameters.


You can configure the running time, rerun properties, and scheduling dependencies of the synchronization task in **Properties**.

i. In the hierarchy tree, click **Data Integration**. Double-click the name of the synchronization task node.

ii. On the right side of the edit page of the synchronization task node, click **Properties** to configure the scheduling parameters. For more information, see [Configure recurrence and dependencies for a node](#).

7. Submit the synchronization task.

After the synchronization task is submitted to the scheduling system, the scheduling system runs the synchronization task at the scheduled time based on the configured scheduling parameters.


i. On the edit page of the synchronization task node, click the  icon.

ii. In the **Commit Node** dialog box, enter your comments in the Change description field.

iii. Click **OK**.

Step 4: View the synchronization task

1. Go to Operation Center.

 **Note** You can also click **Operation Center** in the upper-right corner of the DataStudio console to go to Operation Center.

i. Log on to the [DataWorks console](#) as a project administrator.

ii. Select a region. In the left-side navigation pane, click **Workspaces**.

iii. On the **Workspaces** page, click **Operation Center** in the Actions column that corresponds to the required workspace.

2. In the left-side navigation pane of the Operation Center console, choose **Cycle Task Maintenance > Cycle Task**.
3. On the **Cycle Task** page, view the details about the submitted synchronization task.
 - In the left-side navigation pane, choose **Cycle Task Maintenance > Cycle Instance** to view the task that is scheduled to run on the current date. Click the instance name to view the task running details.
 - You can view logs while a task is running or after the task is completed.

Step 5: View the data exported to OSS

1. Log on to the **OSS console**.
2. Select the corresponding bucket and object name. You can check whether the object contains the content as expected after you download the object.

2.2. Synchronize data from tablestore to MaxCompute


2.2.1. Export full data in script mode

This topic describes how to use the DataWorks console to export full data from Tablestore to MaxCompute.

Step 1: Add a Tablestore data source

To add a Tablestore database as the data source, perform the following steps.

1. Go to the Data Integration homepage.
 - i. Log on to the **DataWorks console** as a project administrator.

 **Note** Only the project administrator role can be used to add data sources. Members who assume other roles can only view data sources.

- ii. Select a region. In the left-side navigation pane, click **Workspaces**.
 - iii. On the **Workspaces** page, click **Data Integration** in the Actions column that corresponds to the required workspace.
2. Add a data source.
 - i. In the Data Integration console, choose **Data Source > Data Sources**.
 - ii. On the **Data Source** page, click **Add data source** in the upper-right corner.
 - iii. In the **Add data source** dialog box, click **OTS** in the **NoSQL** section.
 - iv. In the **Add OTS data source** dialog box, configure the parameters.

Add OTS data source

* Data Source Name :
Custom name

Data source description :

Network connection :
Please Select

type

* Endpoint :

* Table Store instance :

name

* AccessKey ID :

* AccessKey Secret :

Resource Group :
Data Integration
Schedule

If your Data Integration task used this connector, it is necessary to ensure that the connector can be connected by the corresponding resource group. Please refer to the [resource group](#) for detailed concepts and [network solutions](#).

View current best network solution recommendations

Resource group name	Type	Connectivity status (Click status to view details)	Test time	Operation
---------------------	------	---	-----------	-----------

Previous step
Complete

Parameter	Description
Data source	The name of the data source. Example: gps_data.
Description	The description of the data source.
Endpoint	<p>The endpoint of the Tablestore instance. For more information, see Endpoint.</p> <ul style="list-style-type: none"> If the Tablestore instance is in the same region as the MaxCompute project, enter the endpoint to access the Tablestore instance over the classic network. If the Tablestore instance is not in the same region as the MaxCompute project, enter the public endpoint. Do not enter a virtual private cloud (VPC) endpoint.
Table Store instance name	The name of the Tablestore instance.
AccessKey ID	The AccessKey ID and AccessKey secret of your logon account. For more information about how to obtain the AccessKey ID and the AccessKey secret, see Create an AccessKey pair for a RAM user .
AccessKey Secret	

v. Click **Test connectivity** to test the connectivity of the data source.

3. Click **Complete**.

On the **Data Source** page, information about the data source appears.

Step 2: Add a MaxCompute data source

The procedure is similar to that in Step 1, except that in the Add data source dialog box, you must click **MaxCompute** in the **Big Data Storage** section.

In this example, the following figure shows that the data source is named OTS2ODPS.

Add MaxCompute (ODPS) data source

* Data Source Name : OTS2ODPS

Data source description :

Network connection : public

type

* ODPS Endpoint : http://service.odps.aliyun.com/api

Tunnel Endpoint :

* ODPS project name : myh

* AccessKey ID : [REDACTED]

* AccessKey Secret : [REDACTED]

Resource Group : **Data Integration** Schedule

i If your Data Integration task used this connector, it is necessary to ensure that the connector can be connected by the corresponding resource group. Please refer to the [resource group](#) for detailed concepts and [network solutions](#).

[View current best network solution recommendations](#)

Resource group name	Type	Connectivity status (Click status to view)	Last time	Operation

[Previous step](#) [Complete](#)

Step 3: Configure a synchronization task

To create and configure a task to synchronize data from Tablestore to MaxCompute, perform the following steps:


- Go to Data Analytics.
 - Log on to the **DataWorks console** as a project administrator.
 - Select a region. In the left-side navigation pane, click **Workspaces**.
 - On the **Workspaces** page, click **Data Analytics** in the Actions column that corresponds to the workspace.
- On the **Data Analytics** page of the DataStudio console, click **Business Flow** and select a business flow.

For more information about how to create a business flow, see [Create a workflow](#).

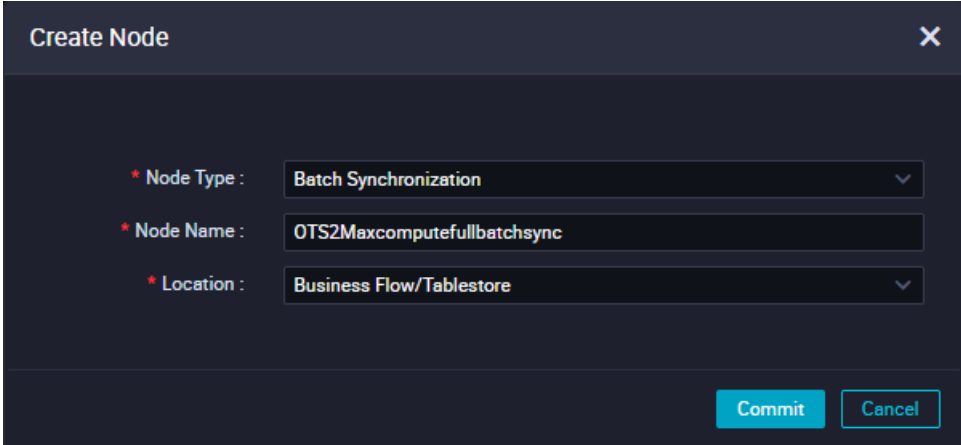
- Create a synchronization task node.

You must create a node for each synchronization task.

- i. Right-click **Data Integration** and then choose **Create > Batch synchronization**.

You can also move the pointer over the  icon, and then choose **Data Integration > Batch synchronization** to create a node.

- ii. In the **Create Node** dialog box, set **Node Name** and **Location**.



The **Create Node** dialog box is shown with the following fields:

- * Node Type :** Batch Synchronization
- * Node Name :** OTS2Maxcomputebatchsync
- * Location :** Business Flow/Tablestore

Buttons at the bottom: **Commit** and **Cancel**.

- iii. Click **Commit**.

4. Configure the Tablestore data source.

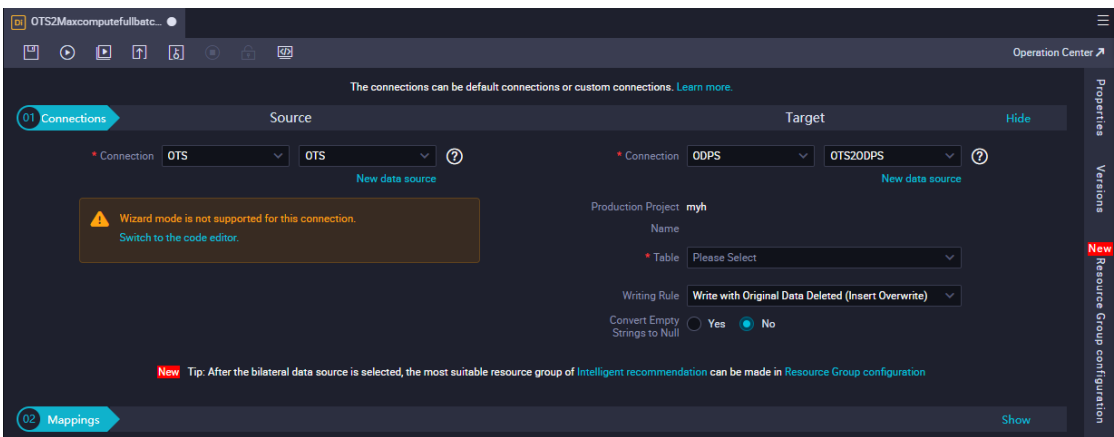
- i. Click **Data Integration**. Double-click the name of the node for the data synchronization task.
- ii. On the edit page of the synchronization task node, configure **Source** and **Target** in the **Connections** section.

■ Configure Source.

Set **Connection** to **OTS** for **Source**.

■ Configure Target.

In the **Target** section, select **ODPS** from the drop-down list next to **Connection**, and set **Table**.



The **Connections** configuration page is shown with the following details:

- Source:**
 - * Connection:** OTS
 - Table:** OTS
 - Writing Rule:** Write with Original Data Deleted (Insert Overwrite)
 - Convert Empty Strings to Null:** No
- Target:**
 - * Connection:** ODPS
 - Table:** OTS2ODPS
 - Production Project:** myh
 - Table:** Please Select
 - Writing Rule:** Write with Original Data Deleted (Insert Overwrite)
 - Convert Empty Strings to Null:** No

A warning message states: "Wizard mode is not supported for this connection. Switch to the code editor."

A tip at the bottom says: "New Tip: After the bilateral data source is selected, the most suitable resource group of Intelligent recommendation can be made in Resource Group configuration"

- iii. Click the  icon or **Switch to the code editor** to configure the script.

Tablestore supports only the script mode to configure the connection. When you use the script to configure the connection, you must configure Tablestore Reader and MaxCompute Writer plug-ins. For more information about specific operations, see [Configure Tablestore Reader](#) and [MaxCompute Writer](#).

On the configuration page of the script, configure the parameters based on the following example:

```
{
  "type": "job",
  "version": "1.0",
  "configuration": {
    "setting": {
      "errorLimit": {
        "record": "0"    # The maximum allowable number of errors that occur.
      },
      "speed": {
        "mbps": "1",    # The maximum amount of traffic. Unit: MB.
        "concurrent": "1" # The number of concurrent threads.
      }
    },
    "reader": {
      "plugin": "ots", # The name of the plug-in used to read data.
      "parameter": {
        "datasource": "", # The name of the data source.
        "table": "", # The name of the data table.
        "column": [ # The names of the columns in Tablestore that need to export to Ma
xCompute.
          {
            "name": "column1"
          },
          {
            "name": "column2"
          },
          {
            "name": "column3"
          },
          {
            "name": "column4"
          },
          {
            "name": "column5"
          }
        ],
        "range": "range": { # The range of data to export. In the full export mode, th
e range is from INF_MIN to INF_MAX.
          "begin": [ # The range of data to export. In full export mode, the range is f
rom INF_MIN to INF_MAX. The number of configuration items in begin must be the same
as the number of primary key columns in the data table in Tablestore.
            {
              "type": "INF_MIN"
            },
            {
              "type": "INF_MIN"
            }
          ]
        }
      }
    }
  }
}
```

```

    {
        "type": "STRING", # The position from which to export data in the third
column starts from begin1.
        "value": "begin1"
    },
    {
        "type": "type": "INT", # The position from which to export data in the f
ourth column starts from 0.
        "value": "0"
    }
],
"end": [ # The position at which data export ends.
    {
        "type": "INF_MAX"
    },
    {
        "type": "INF_MAX"
    },
    {
        "type": "STRING",
        "value": "end1"
    },
    {
        "type": "INT",
        "value": "100"
    }
],
"split": [ # Specify the partition range. Typically, this parameter can be l
eft empty. If the read performance is poor, submit a ticket or join the DingTalk gr
oup 23307953 to contact Tablestore technical support.
    {
        "type": "STRING",
        "value": "splitPoint1"
    },
    {
        "type": "STRING",
        "value": "splitPoint2"
    },
    {
        "type": "STRING",
        "value": "splitPoint3"
    }
]
}
},
"writer": {
    "plugin": "odps", # The name of the plug-in used to write data to MaxCompute.
    "parameter": {
        "datasource": "", # The name of the data source of MaxCompute.
        "column": [], # The names of the columns in MaxCompute. The column names are s
orted in the same order as in Tablestore.
        "table": "", # The name of the table in MaxCompute. The table must be created
before you run the task. Otherwise, the task may fail.
    }
}

```



```

    "partition": "", # This parameter is required if the MaxCompute table is parti
tioned. Do not specify this parameter if the table is not partitioned. The partiti
on to which data is written. The last-level partition must be specified.
    "truncate": false # Specify whether to delete all previous data.
  }
}
}
}

```

You can use the begin and end parameters to configure the range of data to export. For example, a data table contains the pk1 and pk2 primary key columns. The pk1 column is of the STRING type. The pk2 column is of the INTEGER type.

- To export full data from the data table, configure the following parameters:

```

"begin": [ # The position from which data export starts.
  {
    "type": "INF_MIN"
  },
  {
    "type": "INF_MIN"
  }
],
"end": [ # The position at which data export ends.
  {
    "type": "INF_MAX"
  },
  {
    "type": "INF_MAX"
  }
],



```

- To export data from the rows where the value of pk1 is "tablestore", configure the following parameters:

```


"begin": [ # The position from which data export starts.
  {
    "type": "STRING",
    "value": "tablestore"
  },
  {
    "type": "INF_MIN"
  }
],
"end": [ # The position at which data export ends.
  {
    "type": "STRING",
    "value": "tablestore"
  },
  {
    "type": "INF_MAX"
  }
],

```

- iv. Click the  icon to save the data source configurations.
5. Run the synchronization task.
 - i. Click the  icon.
 - ii. In the **Arguments** dialog box, select the resource group for scheduling.
 - iii. Click **OK** to run the task.

After the task is run, you can check whether the task was successful and the number of rows of exported data on the **Runtime Log** tab.
6. Configure the scheduling parameters.


You can configure the running time, rerun properties, and scheduling dependencies of the synchronization task in **Properties**.

 - i. In the hierarchy tree, click **Data Integration**. Double-click the name of the synchronization task node.
 - ii. On the right side of the edit page of the synchronization task node, click **Properties** to configure the scheduling parameters. For more information, see [Configure recurrence and dependencies for a node](#).
7. Submit the synchronization task.
 - i. On the edit page of the synchronization task node, click the  icon.
 - ii. In the **Commit Node** dialog box, enter your comments in the Change description field.
 - iii. Click **OK**.

After the synchronization task is submitted to the scheduling system, the scheduling system runs the synchronization task at the scheduled time based on the configured scheduling parameters.

Step 4: View the synchronization task

1. Go to Operation Center.

 **Note** You can also click **Operation Center** in the upper-right corner of the DataStudio console to go to Operation Center.

- i. Log on to the [DataWorks console](#) as a project administrator.
 - ii. Select a region. In the left-side navigation pane, click **Workspaces**.
 - iii. On the **Workspaces** page, click **Operation Center** in the Actions column that corresponds to the required workspace.
2. In the left-side navigation pane of the Operation Center console, choose **Cycle Task Maintenance > Cycle Task**.
3. On the **Cycle Task** page, view the details about the submitted synchronization task.
 - In the left-side navigation pane, choose **Cycle Task Maintenance > Cycle Instance** to view the task that is scheduled to run on the current date. Click the instance name to view the task running details.
 - You can view logs while a task is running or after the task is completed.

Step 5: View the data imported to MaxCompute

1. Go to the DataMap console.
 - i. Log on to the [DataWorks console](#) as a project administrator.
 - ii. Select a region. In the left-side navigation pane, click **Workspaces**.
 - iii. On the **Workspaces** page, click **Data Map** in the Actions column that corresponds to a workspace.
2. In the top navigation bar of the DataMap console, choose **My Data > Managed by Me**.
3. On the **Managed by Me** tab, click the name of the imported table.
4. On the table details page, click the **Data Preview** tab to view the imported data.