Oracle® NoSQL Database Utilities Guide





Oracle NoSQL Database Utilities Guide, Release 25.3

G12045-06

Copyright © 2024, 2025, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

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

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1 Using Oracle NoSQL Database Migrator

Overview	1
Terminology used with Oracle NoSQL Database Migrator	2
Workflow for Oracle NoSQL Database Migrator	4
Sources and Sinks	17
Supported Sources and Sinks	17
Source and Sink Security	18
Parameters	19
Source Configuration Templates	24
JSON File Source	25
JSON File in OCI Object Storage Bucket	27
MongoDB-Formatted JSON File	29
MongoDB-Formatted JSON File in OCI Object Storage bucket	31
DynamoDB-Formatted JSON File stored in AWS S3	34
DynamoDB-Formatted JSON File	36
Oracle NoSQL Database	38
Oracle NoSQL Database Cloud Service	41
CSV File Source	44
CSV file in OCI Object Storage Bucket	47
Sink Configuration Templates	51
JSON File Sink	51
Parquet File	53
JSON File in OCI Object Storage Bucket	56
Parquet File in OCI Object Storage Bucket	58
Oracle NoSQL Database	61
Oracle NoSQL Database Cloud Service	67
Transformation Configuration Templates	75
ignoreFields	76
includeFields	77
renameFields	77
aggregateFields	78
Mapping of DynamoDB table to Oracle NoSQL table	79
Oracle NoSQL to Parquet Data Type Mapping	80

Mapping of DynamoDB types to Oracle NoSQL types	81
Use Case Demonstrations	82
Migrate from Oracle NoSQL Database to a JSON file	82
Migrate from Oracle NoSQL Database On-Premise to Oracle NoSQL Database Cloud Service	86
Migrate from JSON file source to Oracle NoSQL Database	88
Migrate from MongoDB JSON file to Oracle NoSQL Database	91
Migrate from DynamoDB JSON file to Oracle NoSQL Database	97
Migrate from DynamoDB JSON file in AWS S3 to Oracle NoSQL Database	103
Migrate from CSV file to Oracle NoSQL Database	110
Migrate from Oracle NoSQL Database to OCI Object Storage Using Session Token	
Authentication	114
Troubleshooting the Oracle NoSQL Database Migrator	118

Index

List of Tables

1-1	Migrating TTL metadata	8
1-2	Supported Log Levels for NoSQL Database Migrator	<u>16</u>
1-3	<u>Transformations</u>	<u>76</u>
1-4	Mapping DynamoDB type to Oracle NoSQL type	<u>81</u>
1-5	Migration Failure Causes	118

Using Oracle NoSQL Database Migrator

Learn about Oracle NoSQL Database Migrator and how to use it for data migration.

Oracle NoSQL Database Migrator is a tool that enables you to migrate Oracle NoSQL tables from one data source to another. This tool can operate on tables in Oracle NoSQL Database Cloud Service and Oracle NoSQL Database on-premises and AWS S3. The Migrator tool supports several different data formats and physical media types. Supported data formats are JSON, Parquet, MongoDB-formatted JSON, DynamoDB-formatted JSON, and CSV files. Supported physical media types are files, OCI Object Storage, Oracle NoSQL Database on-premises, Oracle NoSQL Database Cloud Service, and AWS S3.

Topics:

- Overview
- Workflow for Oracle NoSQL Database Migrator
- Supported Sources and Sinks
- Use Case Demonstrations
- Troubleshooting the Oracle NoSQL Database Migrator

Overview

Oracle NoSQL Database Migrator lets you move Oracle NoSQL tables from one data source to another, such as Oracle NoSQL Database on-premises or cloud or even a simple JSON file.

There can be many situations that require you to migrate NoSQL tables *from* or *to* an Oracle NoSQL Database. For instance, a team of developers enhancing a NoSQL Database application may want to test their updated code in the local Oracle NoSQL Database Cloud Service (NDCS) instance using cloudsim. To verify all the possible test cases, they must set up the test data similar to the actual data. To do this, they must copy the NoSQL tables from the production environment to their local NDCS instance, the cloudsim environment. In another situation, NoSQL developers may need to move their application data from on-premise to the cloud and vice-versa, either for development or testing.

In all such cases and many more, you can use Oracle NoSQL Database Migrator to move your NoSQL tables from one data source to another, such as Oracle NoSQL Database on-premise or cloud or even a simple JSON file. You can also copy NoSQL tables from a MongoDB-formatted JSON input file, DynamoDB-formatted JSON input file (either stored in AWS S3 source or from files), or a CSV file into your NoSQL Database on-premises or cloud.

As depicted in the following figure, the NoSQL Database Migrator utility acts as a connector or pipe between the data source and the target (referred to as the sink). In essence, this utility exports data from the selected source and imports that data into the sink. This tool is table-oriented, that is, you can move the data only at the table level. A single migration task operates on a single table and supports migration of table data from source to sink in various data formats.

Oracle NoSQL Database Migrator is designed such that it can support additional sources and sinks in the future. For a list of sources and sinks supported by Oracle NoSQL Database Migrator as of the current release, see Supported Sources and Sinks.





Terminology used with Oracle NoSQL Database Migrator

Learn about the different terms used in the above diagram, in detail.

- **Source:** An entity from where the NoSQL tables are exported for migration. Some examples of sources are Oracle NoSQL Database on-premise or cloud, JSON file, MongoDB-formatted JSON file, DynamoDB-formatted JSON file, and CSV files.
- **Sink:** An entity that imports the NoSQL tables from NoSQL Database Migrator. Some examples for sinks are Oracle NoSQL Database on-premise or cloud and JSON file.

The NoSQL Database Migrator tool supports different types of sources and sinks (that is physical media or repositories of data) and data formats (that is how the data is represented in the source or sink). Supported data formats are JSON, Parquet, MongoDB-formatted JSON, DynamoDB-formatted JSON, and CSV files. Supported source and sink types are files, OCI Object Storage, Oracle NoSQL Database on-premise, and Oracle NoSQL Database Cloud Service.

- Migration Pipe: The data from a source will be transferred to the sink by NoSQL Database Migrator. This can be visualized as a Migration Pipe.
- Transformations: You can add rules to modify the NoSQL table data in the migration pipe.
 These rules are called Transformations. Oracle NoSQL Database Migrator allows data transformations at the top-level fields or columns only. It does not let you transform the data in the nested fields. Some examples of permitted transformations are:
 - Drop or ignore one or more columns,
 - Rename one or more columns, or
 - Aggregate several columns into a single field, typically a JSON field.
- Configuration File: A configuration file is where you define all the parameters required for the migration activity in a JSON format. Later, you pass this configuration file as a single parameter to the runMigrator command from the CLI. A typical configuration file format looks like as shown below.

```
{
  "source": {
    "type" : <source type>,
    //source-configuration for type. See Source Configuration Templates .
},
  "sink": {
    "type" : <sink type>,
    //sink-configuration for type. See Sink Configuration Templates .
},
  "transforms" : {
    //transforms configuration. See Transformation Configuration
Templates .
},
  "migratorVersion" : "<migrator version>",
```



"abortOnError" : <true|false>

Group	Parameters	Mandatory (Y/N)	Purpose	Supported Values
source	type	Y	Represents the source from which to migrate the data. The source provides data and metadata (if any) for migration.	To know the type value for each source, see Supported Sources and Sinks.
source	source- configuration for type	Y	Defines the configuration for the source. These configuration parameters are specific to the type of source selected above.	See Source Configuration Templates . for the complete list of configuration parameters for each source type.
sink	type	Y	Represents the sink to which to migrate the data. The sink is the target or destination for the migration.	To know the type value for each source, see Supported Sources and Sinks.
sink	sink-configuration for type	Y	Defines the configuration for the sink. These configuration parameters are specific to the type of sink selected above.	See Sink Configuration Templates for the complete list of configuration parameters for each sink type.
transforms	transforms configuration	N	Defines the transformations to be applied to the data in the migration pipe.	See Transformation Configuration Templates for the complete list of transformations supported by the NoSQL Data Migrator.
-	migratorVersio n	N	Version of the NoSQL Data Migrator	-



Group	Parameters	Mandatory (Y/N)	Purpose	Supported Values
-	abortOnError	N	Specifies whether to stop the migration activity in case of any error or not.	true, false
			The default value is <i>true</i> indicating that the migration stops whenever it encounters a migration error.	
			If you set this value to false, the migration continues even in case of failed records or other migration errors. The failed records	
			and migration errors will be logged as WARNINGs on the CLI terminal.	

(i) Note

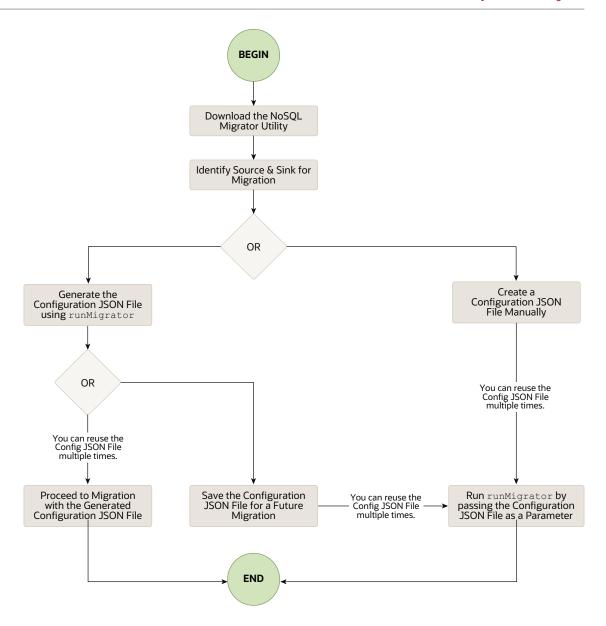
As JSON file is case-sensitive, all the parameters defined in the configuration file are case-sensitive unless specified otherwise.

Workflow for Oracle NoSQL Database Migrator

Learn about the various steps involved in using the Oracle NoSQL Database Migrator utility for migrating your NoSQL data.

The high level flow of tasks involved in using NoSQL Database Migrator is depicted in the below figure.





Download the NoSQL Data Migrator Utility

The Oracle NoSQL Database Migrator utility is available for download from the Oracle NoSQL Downloads page. Once you download and unzip it on your machine, you can access the runMigrator command from the command line interface.



Note

Oracle NoSQL Database Migrator utility requires Java 11 or higher versions to run.

Identify the Source and Sink

Before using the migrator, you must identify the data source and sink. For instance, if you want to migrate a NoSQL table from Oracle NoSQL Database on-premise to a JSON formatted file, your source will be Oracle NoSQL Database and sink will be JSON file. Ensure that the identified source and sink are supported by the Oracle NoSQL Database Migrator by referring



to <u>Supported Sources and Sinks</u>. This is also an appropriate phase to decide the schema for your NoSOL table in the target or sink, and create them.

- Identify Sink Table Schema: If the sink is Oracle NoSQL Database on-premise or cloud, you must identify the schema for the sink table and ensure that the source data matches with the target schema. If required, use transformations to map the source data to the sink table.
 - Default Schema: NoSQL Database Migrator provides an option to create a table with
 the default schema without the need to predefine the schema for the table. This is
 useful primarily when loading JSON source files into Oracle NoSQL Database.
 If the source is a MongoDB-formatted JSON file, the default schema for the table will
 be as follows:

```
CREATE TABLE IF NOT EXISTS  (ID STRING, DOCUMENT JSON, PRIMARY KEY(SHARD(ID))
```

Where:

- tablename = value provided for the table attribute in the configuration.
- ID = _id value from each document of the mongoDB exported JSON source file.
- DOCUMENT = For each document in the mongoDB exported file, the contents excluding the _id field are aggregated into the DOCUMENT column.

If the source is a DynamoDB-formatted JSON file, the default schema for the table will be as follows:

```
CREATE TABLE IF NOT EXISTS <TABLE_NAME>(DDBPartitionKey_name
DDBPartitionKey_type,
[DDBSortKey_name DDBSortKey_type],DOCUMENT JSON,
PRIMARY KEY(SHARD(DDBPartitionKey_name),[DDBSortKey_name]))
```

Where:

- TABLE NAME = value provided for the sink table in the configuration
- DDBPartitionKey name = value provided for the partition key in the configuration
- DDBPartitionKey_type = value provided for the data type of the partition key in the configuration
- DDBSortKey_name = value provided for the sort key in the configuration if any
- DDBSortKey_type = value provided for the data type of the sort key in the configuration if any
- DOCUMENT = All attributes except the partition and sort key of a Dynamo DB table item aggregated into a NoSQL JSON column

If the source format is a CSV file, a default schema is not supported for the target table. You can create a schema file with a table definition containing the same number of columns and data types as the source CSV file. For more details on the Schema file creation, see Providing Table Schema.

For all the other sources, the default schema will be as follows:

CREATE TABLE IF NOT EXISTS <tablename> (ID LONG GENERATED ALWAYS AS IDENTITY, DOCUMENT JSON, PRIMARY KEY(ID))



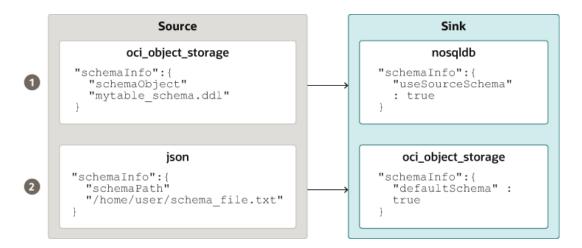
Where:

- tablename = value provided for the table attribute in the configuration.
- ID = An auto-generated LONG value.
- DOCUMENT = The JSON record provided by the source is aggregated into the DOCUMENT column.

(i) Note

If the _id value is not provided as a string in the MongoDB-formatted JSON file, NoSQL Database Migrator converts it into a string before inserting it into the default schema.

- **Providing Table Schema:** NoSQL Database Migrator allows the source to provide schema definitions for the table data using schemaInfo attribute. The schemaInfo attribute is available in all the data sources that do not have an implicit schema already defined. Sink data stores can choose any one of the following options.
 - Use the default schema defined by the NoSQL Database Migrator.
 - Use the source-provided schema.
 - Override the source-provided schema by defining its own schema. For example, if you
 want to transform the data from the source schema to another schema, you need to
 override the source-provided schema and use the transformation capability of the
 NoSQL Database Migrator tool.



The table schema file, for example, mytable_schema.ddl can include table DDL statements. The NoSQL Database Migrator tool executes this table schema file before starting the migration. The migrator tool supports no more than one DDL statement per line in the schema file. For example,

CREATE TABLE IF NOT EXISTS(id INTEGER, name STRING, age INTEGER, PRIMARY KEY(SHARD(ID)))





(i) Note

Migration will fail if the table is present at the sink and the DDL in the schemaPath is different than the table.

Create Sink Table: Once you identify the sink table schema, create the sink table either through the Admin CLI or using the schemaInfo attribute of the sink configuration file. See Sink Configuration Templates .



(i) Note

If the source is a CSV file, create a file with the DDL commands for the schema of the target table. Provide the file path in schemaInfo.schemaPath parameter of the sink configuration file.

Migrating TTL Metadata for Table Rows

Time to Live (TTL) is a mechanism that allows you to automatically expire table rows. TTL is expressed as the amount of time, data is allowed to live in the store. Data that has reached its expiration timeout value can no longer be retrieved, and will not appear in any store statistics.

You can choose to include the TTL metadata for table rows along with the actual data when performing migration of Oracle NoSQL Database tables. The NoSQL Database Migrator provides configuration parameters to support the export and import of table row TTL metadata for the following source types:

Table 1-1	Migrating TTL	_ metadata
-----------	---------------	------------

Source types	Source configuration parameter	Sink configuration parameter
Oracle NoSQL Database	includeTTL	includeTTL
Oracle NoSQL Database Cloud Service	includeTTL	includeTTL
DynamoDB-Formatted JSON File	ttlAttributeName	includeTTL
DynamoDB-Formatted JSON File stored in AWS S3	ttlAttributeName	includeTTL

Exporting TTL metadata in Oracle NoSQL Database and Oracle NoSQL Database Cloud Service

NoSQL Database Migrator provides the includeTTL configuration parameter to support the export of table row's TTL metadata.

When a table is exported, the TTL data is exported for the table rows that have a valid expiration time. If a row does not expire, then the _metadata JSON object is not included explicitly in the exported data because its expiration value is always 0. The NoSQL Database Migrator exports the expiration time for each row as the number of milliseconds since the UNIX epoch (Jan 1st, 1970). For example,

```
//Row 1
    "id" : 1,
    "name" : "xyz",
```



```
"age" : 45,
    " metadata" : {
        "expiration" : 1629709200000
                                      //Row Expiration time in milliseconds
}
//Row 2
    "id" : 2,
    "name" : "abc",
    "age" : 52,
    " metadata" : {
        "expiration": 1629709400000 //Row Expiration time in milliseconds
}
//Row 3 No Metadata for below row as it will not expire
    "id" : 3,
    "name" : "def",
    "age" : 15
}
```

Importing TTL metadata

You can optionally import TTL metadata using the includeTTL configuration parameter in the sink configuration template.

The default reference time of import operation is the current time in milliseconds, obtained from System.currentTimeMillis(), of the machine where the NoSQL Database Migrator tool is running. However, you can also set a custom reference time using the ttlRelativeDate configuration parameter if you want to extend the expiration time and import rows that would otherwise expire immediately. The extension is calculated as follows and added to the expiration time.

```
Extended time = expiration time - reference time
```

The import operation handles the following use cases when migrating table rows containing TTL metadata. These use cases are applicable only when the includeTTL configuration parameter is set to true.

- Use-case 1: No TTL metadata information is present in the importing table row.
 If the row you want to import does not contain TTL information, then the NoSQL Database Migrator sets the TTL=0 for the row.
- Use-case 2: TTL value of the source table row is expired relative to the reference time
 when the table row gets imported.
 - The expired table row is ignored and not written into the store.
- **Use-case 3**: TTL value of the source table row is not expired relative to the reference time when the table row gets imported.
 - The table row gets imported with a TTL value. However, the imported TTL value may not match the original exported TTL value because of the integer hour and day window constraints in the *TimeToLive* class. For example,



Consider an exported table row:

```
{
  "id" : 8,
  "name" : "xyz",
  "_metadata" : {
   "expiration" : 1734566400000 //Thursday, December 19, 2024 12:00:00 AM
in UTC
  }
}
```

The reference time while importing is 1734480000000, which is Wednesday, December 18, 2024 12:00:00 AM.

Imported table row

```
{
  "id" : 8,
  "name" : "xyz",
  "_metadata" : {
    "ttl" : 1734739200000 //Saturday, December 21, 2024 12:00:00 AM
  }
}
```

Importing TTL Metadata in DynamoDB-Formatted JSON File and DynamoDB-Formatted JSON File stored in AWS S3

NoSQL Database Migrator provides an additional configuration parameter, ttlAttributeName to support the import of TTL metadata from the DynamoDB-formatted JSON file items.

DynamoDB exported JSON files include a specific attribute in each item to store the TTL expiration timestamp. To optionally import the TTL values from DynamoDB exported JSON files, you must supply the specific attribute's name as a value to the ttlAttributeName configuration parameter in the DynamoDB-Formatted JSON File or DynamoDB-Formatted JSON File stored in AWS S3 source configuration files. Also, you must set the includeTTL configuration parameter in the sink configuration template. The valid sinks are Oracle NoSQL Database and Oracle NoSQL Database Cloud Service. NoSQL Database Migrator stores TTL information in the _metadata JSON object for the imported item.

The import operation manages the following use cases when migrating table items of the DynamoDB exported JSON files:

• Use case 1: The ttlAttributeName configuration parameter value is set to the TTL attribute name specified in the DynamoDB exported JSON file.

NoSQL Database Migrator imports the expiration time for this item as the number of milliseconds since the UNIX epoch (Jan 1st, 1970).

For example, consider an item in the DynamoDB exported JSON file:

```
{
    "Item": {
        "DeptId": {
            "N": "1"
        },
        "DeptName": {
            "S": "Engineering"
```



```
},
"ttl": {
    "N": "1734616800"
}
}
```

Here, the attribute ttl specifies the time-to-live value for the item. If you set the ttlAttributeName configuration parameter as ttl in the DynamoDB-formatted JSON file or DynamoDB-formatted JSON file stored in AWS S3 source configuration file, NoSQL Database Migrator imports the expiration time for the item as follows:

```
{
    "DeptId": 1,
    "document": {
         "DeptName": "Engineering"
     }
    "_metadata": {
         "expiration": 1734616800000
    }
}
```

(i) Note

You can supply the ttlRelativeDate configuration parameter in the sink configuration template as the reference time for calculating the expiration time.

- Use case 2: The ttlAttributeName configuration parameter value is set, however, the
 value does not exist as an attribute in the item of the DynamoDB exported JSON file.
 NoSQL Database Migrator does not import the TTL metadata information for the given
 item.
- Use case 3: The ttlAttributeName configuration parameter value does not match the
 attribute name in the item of DynamoDB exported JSON file.
 NoSQL Database Migrator handles the import in one of the following ways based on the
 sink configuration:
 - Copies the attribute as a normal field if configured to import using the default schema.
 - Skips the attribute if configured to import using a user-defined schema.

Importing data to a sink with an IDENTITY column

You can import the data from a valid source to a sink table (On-premises/Cloud Services) with an IDENTITY column. You create the IDENTITY column as either GENERATED ALWAYS AS IDENTITY or GENERATED BY DEFAULT AS IDENTITY. For more information on table creation with an IDENTITY column, see Creating Tables With an IDENTITY Column in the *SQL Reference Guide*.

Before importing the data, make sure that the Oracle NoSQL Database table at the sink is empty if it exists. If there is pre-existing data in the sink table, migration can lead to issues such as overwriting existing data in the sink table or skipping source data during the import.

Sink table with IDENTITY column as GENERATED ALWAYS AS IDENTITY



Consider a sink table with the IDENTITY column created as GENERATED ALWAYS AS IDENTITY. The data import is dependent on whether or not the source supplies the values to the IDENTITY column and <code>ignoreFields</code> transformation parameter in the configuration file.

For example, you want to import data from a JSON file source to the Oracle NoSQL Database table as the sink. The schema of the sink table is:

```
CREATE TABLE IF NOT EXISTS migrateID(ID INTEGER GENERATED ALWAYS AS IDENTITY, name STRING, course STRING, PRIMARY KEY

(ID))
```

The Migrator utility handles the data migration as described in the following cases:

Source condition	User action	Migration outcome
CASE 1: Source data does not supply a value for the IDENTITY field of the sink table. Example: JSON source file sample_noID.json	Create/generate the configuration file.	Data migration is successful. IDENTITY column values are auto-generated. Migrated data in Oracle NoSQL Database sink table migrateID: {"ID":1001, "name": "Jane", "cou
<pre>{"name":"John", "course":"Computer Science"} {"name":"Jane", "course":"BioTechnology"} {"name":"Tony", "course":"Electronics"}</pre>		rse":"BioTechnology"} {"ID":1003, "name":"John", "course":"Computer Science"} {"ID":1002, "name":"Tony", "course":"Electronics"}
CASE 2: Source data supplies values for the IDENTITY field of the sink table. Example: JSON source file sampleID. json {"ID":1, "name":"John", "course":"Computer Science"} {"ID":2, "name":"Jane", "course":"BioTechnology"} {"ID":3, "name":"Tony", "course":"Electronics"}	Create/generate the configuration file. You provide an ignoreFields transformation for the ID column in the sink configuration template. "transforms": { "ignoreFields " : ["ID"] }	Data migration is successful. The supplied ID values are skipped and the IDENTITY column values are autogenerated. Migrated data in Oracle NoSQL Database sink table migrateID: {"ID":2003, "name": "John", "course": "Computer Science"} {"ID":2002, "name": "Tony", "course": "Electronics"} {"ID":2001, "name": "Jane", "course": "BioTechnology"}
	You create/ generate the configuration file without the ignoreFields transformation for the IDENTITY column.	Data migration fails with the following error message: "Cannot set value for a generated always identity column".

For more details on the transformation configuration parameters, see the topic Transformation Configuration Templates.

Sink table with IDENTITY column as GENERATED BY DEFAULT AS IDENTITY



Consider a sink table with the IDENTITY column created as GENERATED BY DEFAULT AS IDENTITY. The data import is dependent on whether or not the source supplies the values to the IDENTITY column and <code>ignoreFields</code> transformation parameter.

For example, you want to import data from a JSON file source to the Oracle NoSQL Database table as the sink. The schema of the sink table is:

```
CREATE TABLE IF NOT EXISTS migrateID(ID INTEGER GENERATED BY DEFAULT AS IDENTITY, name STRING, course STRING, PRIMARY KEY (ID))
```

The Migrator utility handles the data migration as described in the following cases:

Source condition	User action	Migration outcome
CASE 1: Source data does not supply a value for the IDENTITY field of the sink table. Example: JSON source file	Create/generate the configuration file.	Data migration is successful. IDENTITY column values are auto-generated. Migrated data in Oracle NoSQL Database sink table migrateID:
sample_noID.json		
{"name":"John",		<pre>{"ID":1,"name":"John","course ":"Computer Science"}</pre>
"course": "Computer Science"}		{"ID":2,"name":"Jane","course
<pre>{"name":"Jane", "course":"BioTechnology"}</pre>		<pre>":"BioTechnology"} {"ID":3,"name":"Tony","course</pre>
{"name":"Tony",		":"Electronics"}
"course": "Electronics" }		
CASE 2: Source data supplies values for the IDENTITY field of the sink table and it is a Primary Key field. Example: JSON source file	Create/generate the configuration file. You provide an ignoreFields	Data migration is successful. The supplied ID values are skipped and the IDENTITY column values are autogenerated.
sampleID.json	transformation for the ID column in	Migrated data in Oracle NoSQL Database sink table migrateID:
<pre>{"ID":1, "name":"John", "course":"Computer Science"} {"ID":2, "name":"Jane", "course":"BioTechnology"} {"ID":3, "name":"Tony", "course":"Electronics"}</pre>	<pre>the sink configuration template (Recommended). "transforms" : { "ignoreFields " : ["ID"] }</pre>	{"ID":1002, "name": "John", "course": "Computer Science"} {"ID":1001, "name": "Jane", "course": "BioTechnology"} {"ID":1003, "name": "Tony", "course": "Electronics"}



Source condition	User action	Migration outcome
	You create/ generate the configuration file without the ignoreFields transformation for the IDENTITY column.	Data migration is successful. The supplied ID values from the source are copied into the ID column in the sink table. When you try to insert an additional row to the table without supplying an ID value, the sequence generator tries to auto-generate the ID value. The sequence generator's starting value is 1.
		As a result, the generated ID value can potentially duplicate one of the existing ID values in the sink table. Since this is a violation of the primary key constraint, an error is returned and the row does not get inserted.
		See Sequence Generator for additional information.
		To avoid the primary key constraint violation, the sequence generator must start the sequence with a value that does not conflict with existing ID values in the sink table. To use the START WITH attribute to make this modification, see the example below:
		Example : Migrated data in Oracle NoSQL Database sink table migrateID:
		<pre>{"ID":1, "name":"John", "course ":"Computer Science"} {"ID":2, "name":"Jane", "course ":"BioTechnology"} {"ID":3, "name":"Tony", "course ":"Electronics"}</pre>
		To find the appropriate value for the sequence generator to insert in the ID column, fetch the maximum value of the ID field using the following query:
		SELECT max(ID) FROM migrateID
		Output:
		{"Column_1":3}
	_	The maximum value of the ID column in the sink table is 3. You want the sequence generator to start generating the ID values beyond 3 to avoid duplication. You update the sequence



Source condition	User action	Migration outcome
		generator's START WITH attribute to 4 using the following statement:
		ALTER Table migrateID (MODIFY ID GENERATED BY DEFAULT AS IDENTITY (START WITH 4))
		This will start the sequence at 4.
		Now when you insert rows to the sink table without supplying the ID values, the sequence generator auto-generates the ID values from 4 onwards averting the duplication of the IDs.

For more details on the transformation configuration parameters, see the topic Transformation Configuration Templates.

Run the runMigrator command

The runMigrator executable file is available in the extracted NoSQL Database Migrator files. You must install Java 11 or higher version and bash on your system to successfully run the runMigrator command.

You can run the runMigrator command in two ways:

1. By creating the configuration file using the runtime options of the runMigrator command as shown below.

```
[~]$ ./runMigrator
configuration file is not provided. Do you want to generate configuration?
(y/n)
[n]: y
...
...
```

- When you invoke the runMigrator utility, it provides a series of run time options and creates the configuration file based on your choices for each option.
- After the utility creates the configuration file, you have a choice to either proceed with the migration activity in the same run or save the configuration file for a future migration.
- Irrespective of your decision to proceed or defer the migration activity with the generated configuration file, the file will be available for edits or customization to meet your future requirements. You can use the customized configuration file for migration later.
- 2. By passing a manually created configuration file (in the JSON format) as a runtime parameter using the -c or --config option. You must create the configuration file manually



before running the runMigrator command with the -c or --config option. For any help with the source and sink configuration parameters, see Sources and Sinks.

[~]\$./runMigrator -c </path/to/the/configuration/json/file>

Logging Migrator Progress

NoSQL Database Migrator tool provides options, which enables trace, debugging, and progress messages to be printed to standard output or to a file. This option can be useful in tracking the progress of migration operation, particularly for very large tables or data sets.

Log Levels

To control the logging behavior through the NoSQL Database Migrator tool, pass the --log-level or -l run time parameter to the runMigrator command. You can specify the amount of log information to write by passing the appropriate log level value.

\$./runMigrator --log-level <loglevel>

Example:

\$./runMigrator --log-level debug

Table 1-2 Supported Log Levels for NoSQL Database Migrator

Log Level	Description
warning	Prints errors and warnings.
info (default)	Prints the progress status of data migration such as validating source, validating sink, creating tables, and count of number of data records migrated.
debug	Prints additional debug information.
all	Prints everything. This level turns on all levels of logging.

Log File:

You can specify the name of the log file using --log-file or -f parameter. If --log-file is passed as run time parameter to the runMigrator command, the NoSQL Database Migrator writes all the log messages to the file else to the standard output.

\$./runMigrator --log-file <log file name>

Example:

\$./runMigrator --log-file nosql_migrator.log

Limitation

Oracle NoSQL Database Migrator does not lock the database during backup and block other users. Therefore, it is highly recommended not to perform the following activities when a migration task is running:

- Any DML/DDL operations on the source table.
- · Any topology-related modification on the data store.



Sources and Sinks

Learn about the different sources and sinks supported by the Oracle NoSQL Database Migrator utility and their configuration templates.

Topics:

- Supported Sources and Sinks
- Source and Sink Security
- Parameters
- Source Configuration Templates
- Sink Configuration Templates
- Transformation Configuration Templates
- Mapping of DynamoDB table to Oracle NoSQL table
- Oracle NoSQL to Parquet Data Type Mapping
- Mapping of DynamoDB types to Oracle NoSQL types

Supported Sources and Sinks

This topic provides the list of the sources and sinks supported by the Oracle NoSQL Database Migrator.

You can use any combination of a valid source and sink from this table for the migration activity. However, you must ensure that at least one of the ends, that is, source or sink must be an Oracle NoSQL product. You can not use the NoSQL Database Migrator to move the NoSQL table data from one file to another.

Type (value)	Format (value)	Valid Source	Valid Sink
Oracle NoSQL Database (nosqldb)	NA	Y	Y
Oracle NoSQL Database Cloud Service (nosqldb_cloud)	NA	Y	Y
File system (file)	JSON (json)	Y	Y
	MongoDB JSON (mongodb_json)	Y	N
	DynamoDB JSON (dynamodb_json)	Y	N



Type (value)	Format (value)	Valid Source	Valid Sink
	Parquet(parquet)	N	Y
	CSV (csv)	Y	N
OCI Object Storage (object_storage_oci)	JSON (json)	Y	Y
	MongoDB JSON (mongodb_json)	Y	N
	Parquet(parquet)	N	Y
	CSV (csv)	Y	N
AWS S3	DynamoDB JSON (dynamodb_json)	Y	N

Note

Many configuration parameters are common across the source and sink configuration. For ease of reference, the description for such parameters is repeated for each source and sink in the documentation sections, which explain configuration file formats for various types of sources and sinks. In all the cases, the syntax and semantics of the

Source and Sink Security

Some of the source and sink types have optional or mandatory security information for authentication purposes.

parameters with the same name are identical.

All sources and sinks that use services in the Oracle Cloud Infrastructure (OCI) can use certain parameters for providing optional security information. This information can be provided using an OCI configuration file or Instance Principal.

Oracle NoSQL Database sources and sinks require mandatory security information if the installation is secure and uses an Oracle Wallet-based authentication. This information can be provided by adding a jar file to the <migrator_home>/lib directory.

Wallet-based Authentication

If an Oracle NoSQL Database installation uses Oracle Wallet-based authentication, you must include additional jar files that are a part of the EE installation. For more information, see Oracle Wallet.



Without the jar files, you will get the following error message:

Could not find kvstore-ee.jar and kvstore-ee-<version>.jar in lib directory. Copy kvstore-ee.jar and kvstore-ee-<version>.jar to lib directory

To prevent the exception shown above, you must copy kvstore-ee.jar and kvstore-ee-<version>. jar files from your EE server package to the <MIGRATOR HOME>/lib directory. <MIGRATOR_HOME> is the nosql-migrator-M.N.O/ directory created by extracting the Oracle NoSQL Database Migrator package and M.N.O represent the software release.major.minor numbers. For example, nosql-migrator-1.1.0/lib.



(i) Note

The wallet-based authentication is supported ONLY in the Enterprise Edition (EE) of Oracle NoSQL Database.

Authenticating with Instance Principals

Instance principals is an IAM service feature that enables instances to be authorized actors (or principals) that can perform actions on service resources. Each compute instance has its own identity, and it authenticates using the certificates added to it.

Oracle NoSQL Database Migrator provides an option to connect to a NoSQL cloud and OCI Object Storage sources and sinks using instance principal authentication. It is only supported when the NoSQL Database Migrator tool is used within an OCI compute instance, for example, the NoSQL Database Migrator tool running in a VM hosted on OCI. To enable this feature use the useInstancePrincipal attribute of the NoSQL cloud source and sink configuration file. For more information on configuration parameters for different types of sources and sinks, see Source Configuration Templates and Sink Configuration Templates.

For more information on instance principals, see Calling Services from an Instance.

Parameters

The NoSQL Database Migrator requires a configuration file where you define all the parameters to perform the migration activity. A few parameters are common across several sources and sinks. This topic provides a list of these common parameters. For the list of other parameters that are unique to individual sources or sinks, see the corresponding configuration template sections.

Common Configuration Parameters

The following are the common configuration parameters. See the individual configuration template sections for examples.

bucket

Purpose: Specifies the name of the OCI Object Storage bucket, which contains the source/sink objects.

Ensure that the required bucket already exists in the OCI Object Storage instance and has read/write permissions.

Data Type: string Mandatory (Y/N): Y



chunkSize

Purpose: Specifies the maximum size of a chunk of table data to be stored at the sink. The value is in MB. During migration, a table is split into chunkSize chunks and each chunk is written as a separate file to the sink. A new file is created when the source data that is being migrated exceeds the chunkSize value.

If not specified, defaults to 32MB. The valid value is an integer between 1 to 1024.

For details on how to improve the migration speed using the chunkSize parameter, see **Best Practices.**

Data Type: integer Mandatory (Y/N): N

credentials

Purpose: Specifies the absolute path to a file containing OCI credentials. The NoSQL Database Migrator uses this file to connect to the OCI service such as Oracle NoSQL Database Cloud Service, OCI Object Storage, and so on.

The default value is \$HOME/.oci/config

See Example Configuration for an example of the credentials file.



(i) Note

You can select only one of the authentication options. Therefore, specify only one of these parameters - credentials, useInstancePrincipal, useDelegationToken, useSessionToken, or useOKEWorkloadIdentity in the configuration template.

- Data Type: string
- Mandatory (Y/N): N

credentialsProfile

Purpose: Specifies the name of the configuration profile to be used to connect to the OCI service such as Oracle NoSQL Database Cloud Service, OCI Object Storage, and so on. User account credentials are referred to as a profile.

If you do not specify this value, the NoSQL Database Migrator uses the DEFAULT profile.



Note

This parameter is valid only if the <u>credentials</u> parameter is specified.

- Data Type: string
- Mandatory (Y/N): N

endpoint

- **Purpose:** Specifies one of the following:
 - The Service endpoint URL or the Region ID for the OCI Object Storage service. For the list of OCI Object Storage service endpoints, see Object Storage Endpoints.



 The Service endpoint URL or the Region ID for the Oracle NoSQL Database Cloud Service.

You can either specify the complete URL or the Region ID alone. For the list of data regions supported for Oracle NoSQL Database Cloud Service, see Data Regions and Associated Service URLs in the *Oracle NoSQL Database Cloud Service* document.

• Data Type: string

Mandatory (Y/N): Y

format

Purpose: Specifies the source/sink format.

Data Type: stringMandatory (Y/N): Y

namespace

Purpose: Specifies the namespace of the OCI Object Storage service. This is an optional
parameter. If you don't specify this parameter, the Migrator utility uses the namespace
assigned to the tenancy.

For example, the namespace parameter is helpful when you want to use an OCI OS from a tenancy that is different from yours. In such cases, the OCI OS tenancy's namespace is different from your tenancy's namespace. During migration, the Migrator utility defaults to your tenancy's namespace unless specified otherwise. Therefore, to direct the Migrator utility to pick OCI OS tenancy's namespace, you must specify OCI OS tenancy's name in the namespace parameter.

Data Type: string

Mandatory (Y/N): N

prefix

- Purpose: The prefix acts as a logical container or directory for storing data in the OCI Object Storage bucket.
 - Source configuration template: If the prefix parameter in specified, all the objects from the directory named in the prefix parameter are migrated. Else, all the objects present in the bucket are migrated.
 - Sink configuration template: If the prefix parameter is specified, a directory with the given prefix is created in the bucket and the objects are migrated into this directory. Else, the table name from the source is used as the prefix. If any object with the same name already exists in the bucket, it is overwritten.

For more information about prefixes, see Object Naming Using Prefixes and Hierarchies.

Data Type: string

Mandatory (Y/N): N

requestTimeoutMs

 Purpose: Specifies the time to wait for each read/write operation from/to the store to complete. This is provided in milliseconds. The default value is 5000. The value can be any positive integer.

Data Type: integer

Mandatory (Y/N): N



security

Purpose: Specifies the absolute path to the security login file that contains your store
credentials if your store is a secure store. For more information about the security login file,
see Configuring Security with Remote Access in the Administrator's Guide.

You can use either password file based authentication or wallet based authentication. However, the wallet based authentication is supported only in the Enterprise Edition (EE) of Oracle NoSQL Database. For more information on wallet-based authentication, see Source and Sink Security.

The Community Edition(CE) edition supports password file based authentication only.

- Data Type: string
- Mandatory (Y/N): Y, for a secure store

type

Purpose: Identifies the source/sink type.

Data Type: string

Mandatory (Y/N): Y

useDelegationToken

Purpose: Specifies whether or not the NoSQL Database Migrator tool uses a delegation token authentication to connect to the OCI services. You must use the delegation token authentication to run the Migrator utility from the Cloud Shell. The delegation token is automatically created for the user when the Cloud Shell is invoked.

The default value is false.

• Data Type: boolean

Mandatory (Y/N): N



Note

- The authentication with delegation token is supported only when the NoSQL Database Migrator tool is running from a Cloud Shell.
- You can select only one of the authentication options. Therefore, specify only
 one of these parameters <u>credentials</u>, <u>useInstancePrincipal</u>,
 useDelegationToken, <u>useSessionToken</u>, or <u>useOKEWorkloadIdentity</u> in
 the configuration template.
- The Cloud Shell supports migration only between the following sources and sinks:

Туре	Valid source	Valid sink	
Oracle NoSQL Database Cloud Service	Υ	Y	
(nosqldb_cloud)			
File (JSON file in the home directory)	Υ	Υ	
OCI Object Storage (JSON file)	Υ	Υ	
<pre>(object_storage_oci)</pre>			
OCI Object Storage (Parquet file)	N	Υ	
<pre>(object_storage_oci)</pre>			

useInstancePrincipal

Purpose: Specifies whether or not the NoSQL Database Migrator tool uses instance
principal authentication to connect to the OCI service such as Oracle NoSQL Database
Cloud Service, OCI Object Storage, and so on. For more information on Instance Principal
authentication method, see Source and Sink Security.

The default value is false.

Note

- The authentication with Instance Principals is supported only when the NoSQL Database Migrator tool is running within an OCI compute instance, for example, the NoSQL Database Migrator tool running in a VM hosted on OCI.
- You can select only one of the authentication options. Therefore, specify only one of these parameters <u>credentials</u>, useInstancePrincipal, <u>useDelegationToken</u>, <u>useSessionToken</u>, or <u>useOKEWorkloadIdentity</u> in the configuration template.

• Data Type: boolean

Mandatory (Y/N): N



useOKEWorkloadIdentity

 Purpose: Specifies whether or not the NoSQL Database Migrator tool uses Workload Identity Authentication (WIA) to access OCI Object Storage and Oracle NoSQL Database Cloud Service from an Oracle Kubernetes Engine (OKE) pod. The default value is false.

Data Type: booleanMandatory (Y/N): N

Note

You can select only one of the authentication options. Therefore, specify only one of these parameters - credentials, useDelegationToken, <a href="mailto:useDelegationToken"

useSessionToken

- Purpose: Specifies whether or not the NoSQL Database Migrator tool uses a session token authentication to connect to OCI services such as OCI Object Storage (OCI OS) and Oracle NoSQL Database Cloud Service. The default value is false.
- Data Type: boolean
- Mandatory (Y/N): N

To use the session token-based authentication, you must generate a session token using OCI Command Line Interface (CLI) commands.. For a sample use case, see Migrate from Oracle NoSQL Database to OCI Object Storage Using Session Token Authentication.

(i) Note

- While using session token authentication, you must specify the path to the OCI config file in the credentials parameter and the profile used while generating the session token in the credentialsProfile parameter. If you don't set the credentials parameter in configuration template, the Migrator utility looks for the credentials file in the path \$HOME/.oci. If you don't set the credentialsProfile parameter in configuration template, the Migrator utility uses the default profile name (DEFAULT) from the OCI config file.
 - If the Migrator utility is unable to find the credentials file, the migration fails with an error message conveying the non-existence of the OCI credential file.
- You can select only one of the authentication options. Therefore, specify only one
 of these parameters <u>credentials</u>, <u>useInstancePrincipal</u>, <u>useDelegationToken</u>,
 useSessionToken, or <u>useOKEWorkloadIdentity</u> in the configuration template.

Source Configuration Templates

Learn about the source configuration file formats for each valid source and the purpose of each configuration parameter.

For the configuration file template, see **Configuration File** in Terminology used with Oracle NoSQL Database Migrator.



For details on valid sink formats for each of the source, see Sink Configuration Templates.

Topics

The following topics describe the source configuration templates referred by Oracle NoSQL Database Migrator to copy the data from the given source to a valid sink.

- JSON File Source
 - Specified file or directory containing the JSON data.
- JSON File in OCI Object Storage Bucket
 Specified JSON file in the OCI Object Storage bucket.
- MongoDB-Formatted JSON File
 - Specified file or directory containing the MongoDB formatted JSON data.
- MongoDB-Formatted JSON File in OCI Object Storage bucket
 Specified MongoDB exported JSON file stored in the OCI Object Storage bucket.
- DynamoDB-Formatted JSON File stored in AWS S3
 Specified DynamoDB exported JSON file stored in the AWS S3 storage.
- <u>DynamoDB-Formatted JSON File</u>
 Specified DynamoDB exported JSON file from a file system.
- Oracle NoSQL Database
 Specified table in Oracle NoSQL Database.
- Oracle NoSQL Database Cloud Service
 Specified table in Oracle NoSQL Database Cloud Service.
- <u>CSV File Source</u>
 Specified file or directory containing the CSV data.
- CSV file in OCI Object Storage Bucket
 Specified CSV file in the OCI Object Storage bucket.

JSON File Source

The configuration file format for JSON file as a source of NoSQL Database Migrator is shown below.

You can migrate a JSON source file by specifying the file path or a directory in the source configuration template.

A sample JSON source file is as follows:

```
{"id":6,"val_json":{"array":
["q","r","s"],"date":"2023-02-04T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]],"nested":{"arrayofobjects":
[{"datefield":"2023-02-04T02:38:57.520Z","numfield":30,"strfield":"foo54"},
{"datefield":"2023-02-04T02:38:57.520Z","numfield":56,"strfield":"bar23"}],"ne
stNum":10,"nestString":"bar"},"num":1,"string":"foo"}}
{"id":3,"val_json":{"array":
["g","h","i"],"date":"2023-02-02T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]],"nested":{"arrayofobjects":
[{"datefield":"2023-02-04T02:38:57.520Z","numfield":28,"strfield":"foo3"},
{"datefield":"2023-02-04T02:38:57.520Z","numfield":38,"strfield":"bar"}],"nest
Num":10,"nestString":"bar"},"num":1,"string":"foo"}}
```



Source Configuration Template

```
"source": {
  "type": "file",
  "format": "json",
  "dataPath": "<path/to/JSON/[file|dir]>",
  "schemaInfo": {
      "schemaPath": "<path/to/schema/file>"
    }
},
```

Source Parameters

Common Configuration Parameters

```
type
Use "type" : "file"format
Use "format" : "json"
```

Unique Configuration Parameters

- dataPath
- schemalnfo
- · schemaInfo.schemaPath

dataPath

 Purpose: Specifies the absolute path to a file or directory containing the JSON data for migration.

You must ensure that this data matches with the NoSQL table schema defined at the sink. If you specify a directory, the NoSQL Database Migrator identifies all the files with the .json extension in that directory for the migration. Sub-directories are not supported.

- Data Type: string
- Mandatory (Y/N): Y
- Example:
 - Specifying a JSON file

```
"dataPath" : "/home/user/sample.json"
```

Specifying a directory

```
"dataPath" : "/home/user"
```

schemalnfo

- Purpose: Specifies the schema of the source data being migrated. This schema is passed to the NoSQL sink.
- Data Type: Object
- Mandatory (Y/N): N



schemalnfo.schemaPath

- Purpose: Specifies the absolute path to the schema definition file containing DDL statements for the NoSQL table being migrated.
- Data Type: string
- Mandatory (Y/N): Y
- Example:

```
"schemaInfo": {
   "schemaPath": "<path to the schema file>"
}
```

JSON File in OCI Object Storage Bucket

The configuration file format for JSON file in OCI Object Storage bucket as a source of NoSQL Database Migrator is shown below.

You can migrate a JSON file in the OCI Object Storage bucket by specifying the name of the bucket in the source configuration template.

A sample JSON source file in the OCI Object Storage bucket is as follows:

```
{"id":6,"val_json":{"array":
["q","r","s"],"date":"2023-02-04T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]],"nested":{"arrayofobjects":
[{"datefield":"2023-02-04T02:38:57.520Z","numfield":30,"strfield":"foo54"},
{"datefield":"2023-02-04T02:38:57.520Z","numfield":56,"strfield":"bar23"}],"ne
stNum":10,"nestString":"bar"},"num":1,"string":"foo"}}
{"id":3,"val_json":{"array":
["g","h","i"],"date":"2023-02-04T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]],"nested":{"arrayofobjects":
[{"datefield":"2023-02-04T02:38:57.520Z","numfield":28,"strfield":"foo3"},
{"datefield":"2023-02-04T02:38:57.520Z","numfield":38,"strfield":"bar"}],"nest
Num":10,"nestString":"bar"},"num":1,"string":"foo"}}
```

Note

The valid sink types for OCI Object Storage source type are nosqldb and nosqldb_cloud.

Source Configuration Template

```
"source" : {
  "type" : "object_storage_oci",
  "format" : "json",
  "endpoint" : "<OCI Object Storage service endpoint URL or region ID>",
  "namespace" : "<OCI Object Storage namespace>",
  "bucket" : "<bucket name>",
  "prefix" : "<object prefix>",
  "schemaInfo" : {
      "schemaObject" : "<object name>"
```



```
},
"credentials": "</path/to/oci/config/file>",
"credentialsProfile": "<profile name in oci config file>",
"useInstancePrincipal": <true | false>,
"useDelegationToken": <true | false>,
"useSessionToken": <true | false>,
"useOKEWorkloadIdentity": <true | false>)
}
```

Source Parameters

Common Configuration Parameters

```
type
Use "type" : "object_storage_oci"
Use "format" : "json"
endpoint
Example:
   Region ID: "endpoint" : "us-ashburn-1"
   URL format: "endpoint" : "https://objectstorage.us-
   ashburn-1.oraclecloud.com"
namespace
Example: "namespace" : "my-namespace"
bucket
Example: "bucket" : "my-bucket"
prefix
Example:
1. "prefix" : "my_table/Data/000000.json" (migrates only 000000.json)
2. "prefix" : "my_table/Data" (migrates all the objects with prefix my_table/Data)
credentials
Example:
1. "credentials" : "/home/user/.oci/config"
2. "credentials" : "/home/user/security/config"
<u>credentialsProfile</u>
Example:
1. "credentialsProfile" : "DEFAULT"
2. "credentialsProfile" : "ADMIN USER"
useInstancePrincipal
Example: "useInstancePrincipal" : true
useDelegationToken
Example: "useDelegationToken" : true
```





(i) Note

The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

useSessionToken

Example: "useSessionToken" : true

Unique Configuration Parameters

- schemalnfo
- schemaInfo.schemaObject

schemalnfo

- Purpose: Specifies the schema of the source data being migrated. This schema is passed to the NoSQL sink.
- Data Type: Object
- Mandatory (Y/N): N

schemalnfo.schemaObject

- Purpose: Specifies the name of the object in the bucket where NoSQL table schema definitions for the data being migrated are stored.
- Data Type: string
- Mandatory (Y/N): Y
- **Example:**

```
"schemaInfo": {
  "schemaObject": "mytable/Schema/schema.ddl"
```

MongoDB-Formatted JSON File

The configuration file format for MongoDB-formatted JSON File as a source of NoSQL Database Migrator is shown below.

You can migrate a MongoDB exported JSON data by specifying the file or directory in the source configuration template.

MongoDB supports two types of extensions to the JSON format of files, Canonical mode and Relaxed mode. You can supply the MongoDB-formatted JSON file that is generated using the mongoexport tool in either Canonical or Relaxed mode. Both the modes are supported by the NoSQL Database Migrator for migration.

For more information on the MongoDB Extended JSON (v2) file, See mongoexport formats.

For more information on the generation of MongoDB-formatted JSON file, see mongoexport for more information.



A sample MongoDB-formatted Relaxed mode JSON file is as follows:

```
{"_id":0,"name":"Aimee Zank","scores":
[{"score":1.463179736705023,"type":"exam"},
{"score":11.78273309957772,"type":"quiz"},
{"score":35.8740349954354,"type":"homework"}]}
{"_id":1, "name": "Aurelia Menendez", "scores":
[{"score":60.06045071030959,"type":"exam"},
{"score":52.79790691903873,"type":"quiz"},
{"score":71.76133439165544,"type":"homework"}]}
{" id":2,"name":"Corliss Zuk","scores":
[{"score":67.03077096065002,"type":"exam"},
{"score":6.301851677835235,"type":"quiz"},
{"score":66.28344683278382,"type":"homework"}]}
{"_id":3,"name":"Bao Ziglar","scores":
[{"score":71.64343899778332,"type":"exam"},
{"score":24.80221293650313,"type":"quiz"},
{"score":42.26147058804812,"type":"homework"}]}
{"_id":4, "name": "Zachary Langlais", "scores":
[{"score":78.68385091304332,"type":"exam"},
{"score":90.2963101368042,"type":"quiz"},
{ "score": 34.41620148042529, "type": "homework" } ] }
```

Source Configuration Template

```
"source": {
  "type": "file",
  "format": "mongodb_json",
  "dataPath": "</path/to/json/[file|dir]>",
  "schemaInfo": {
     "schemaPath": "</path/to/schema/file>"
  }
}
```

Source Parameters

Common Configuration Parameters

```
type
Use "type" : "file"

format
Use "format" : "mongodb_json"
```

Unique Configuration Parameters

- dataPath
- schemaInfo
- schemalnfo.schemaPath

dataPath

 Purpose: Specifies the absolute path to a file or directory containing the MongoDB exported JSON data for migration.

You can supply the MongoDB-formatted JSON file that is generated using the mongoexport tool.



If you specify a directory, the NoSQL Database Migrator identifies all the files with the .json extension in that directory for the migration. Sub-directories are not supported. You must ensure that this data matches with the NoSQL table schema defined at the sink.

- Data Type: string
- Mandatory (Y/N): Y
- Example:
 - Specifying a MongoDB formatted JSON file

```
"dataPath" : "/home/user/sample.json"
```

Specifying a directory

```
"dataPath" : "/home/user"
```

schemalnfo

- Purpose: Specifies the schema of the source data being migrated. This schema is passed to the valid sink.
- Data Type: Object
- Mandatory (Y/N): N

schemalnfo.schemaPath

- Purpose: Specifies the absolute path to the schema definition file containing DDL statements for the NoSQL table being migrated.
- Data Type: string
- Mandatory (Y/N): Y
- Example:

```
"schemaInfo" : {
   "schemaPath" : "/home/user/mytable/Schema/schema.ddl"
}
```

MongoDB-Formatted JSON File in OCI Object Storage bucket

The configuration file format for MongoDB-Formatted JSON file in OCI Object Storage bucket as a source of NoSQL Database Migrator is shown below.

You can migrate the MongoDB exported JSON data in the OCI Object Storage bucket by specifying the name of the bucket in the source configuration template.

Extract the data from MongoDB using the *mongoexport* utility and upload it to the OCI Object Storage bucket. See mongoexport for more information. MongoDB supports two types of extensions to the JSON format of files, *Canonical mode* and *Relaxed mode*. Both formats are supported in the OCI Object Storage bucket.

A sample MongoDB-formatted Relaxed mode JSON File is as follows:

```
{"_id":0,"name":"Aimee Zank","scores":
[{"score":1.463179736705023,"type":"exam"},
{"score":11.78273309957772,"type":"quiz"},
{"score":35.8740349954354,"type":"homework"}]}
{"_id":1,"name":"Aurelia Menendez","scores":
[{"score":60.06045071030959,"type":"exam"},
```



```
{"score":52.79790691903873,"type":"quiz"},
{"score":71.76133439165544,"type":"homework"}]}
{"_id":2,"name":"Corliss Zuk","scores":
[{"score":67.03077096065002,"type":"exam"},
{"score":6.301851677835235,"type":"quiz"},
{"score":66.28344683278382,"type":"homework"}]}
{"_id":3,"name":"Bao Ziglar","scores":
[{"score":71.64343899778332,"type":"exam"},
{"score":24.80221293650313,"type":"quiz"},
{"score":42.26147058804812,"type":"homework"}]}
{"_id":4,"name":"Zachary Langlais","scores":
[{"score":78.68385091304332,"type":"exam"},
{"score":90.2963101368042,"type":"quiz"},
{"score":34.41620148042529,"type":"homework"}]}
```

(i) Note

The valid sink types for OCI Object Storage source type are nosqldb and nosqldb_cloud.

Source Configuration Template

```
"source" : {
  "type" : "object_storage_oci",
  "format" : "mongodb_json",
  "endpoint" : "<OCI Object Storage service endpoint URL or region ID>",
  "namespace" : "<OCI Object Storage namespace>",
  "bucket" : "<bucket name>",
  "prefix" : "<object prefix>",
  "schemaInfo" : {
      "schemaObject" : "<object name>"
  },
  "credentials" : "</path/to/oci/config/file>",
  "credentialsProfile" : "<profile name in oci config file>",
  "useInstancePrincipal" : <true|false>,
  "useSessionToken" : <true|false>,
  "useOKEWorkloadIdentity" : <true|false>
}
```

Source Parameters

Common Configuration Parameters

```
    type
        Use "type" : "object_storage_oci"
    format
        Use "format" : "mongodb_json"
    endpoint
        Example:

            Region ID: "endpoint" : "us-ashburn-1"
```



```
URL format: "endpoint" : "https://objectstorage.us-ashburn-1.oraclecloud.com"
```

namespace

Example: "namespace" : "my-namespace"

bucket

Example: "bucket" : "my-bucket"

prefix

Example:

- 1. "prefix" : "mongo_export/Data/table.json" (migrates only table.json)
- 2. "prefix" : "mongo_export/Data" (migrates all the objects with prefix mongo_export/
 Data)

Note

If you do not provide any value, all the objects present in the bucket are migrated.

credentials

Example:

- 1. "credentials" : "/home/user/.oci/config"
- 2. "credentials" : "/home/user/security/config"
- credentialsProfile

Example:

- 1. "credentialsProfile" : "DEFAULT"
- 2. "credentialsProfile" : "ADMIN_USER"
- <u>useInstancePrincipal</u>

Example: "useInstancePrincipal" : true

• <u>useDelegationToken</u>

Example: "useDelegationToken" : true



The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

useSessionToken

Example: "useSessionToken" : true

Unique Configuration Parameters

- schemalnfo
- schemalnfo.schemaObject

schemalnfo

 Purpose: Specifies the schema of the source data being migrated. This schema is passed to the NoSQL sink.



- Data Type: Object
- Mandatory (Y/N): N

schemalnfo.schemaObject

- Purpose: Specifies the name of the object in the bucket where NoSQL table schema definitions for the data being migrated are stored.
- Data Type: string
- Mandatory (Y/N): Y
- Example:

```
"schemaInfo": {
   "schemaObject": "mytable/Schema/schema.ddl"
}
```

DynamoDB-Formatted JSON File stored in AWS S3

The configuration file format for DynamoDB-formatted JSON File in AWS S3 as a source of NoSQL Database Migrator is shown below.

You can migrate a file containing the DynamoDB exported JSON data from the AWS S3 storage by specifying the path in the source configuration template.

A sample DynamoDB-formatted JSON File is as follows:

You must export the DynamoDB table to AWS S3 storage as specified in <u>Exporting DynamoDB</u> table data to Amazon S3.

The valid sink types for DynamoDB-formatted JSON stored in AWS S3 are nosqldb and nosqldb_cloud.

Source Configuration Template

```
"source" : {
  "type" : "aws_s3",
  "format" : "dynamodb_json",
  "ttlAttributeName" : <DynamoDB exported TTL attribute name>,
  "s3URL" : "<S3 object url>",
  "credentials" : "</path/to/aws/credentials/file>",
```



```
"credentialsProfile" : <"profile name in aws credentials file">
}
```

Source Parameters

Common Configuration Parameters

Use "format" : "dynamodb_json"

type Use "type" : "aws_s3"format



If the value of the type parameter is aws_s3, then the format must be dynamodb_json.

Unique Configuration Parameters

- s3URL
- credentials
- credentialsProfile
- ttlAttributeName

s3URL

• Purpose: Specifies the URL of an exported DynamoDB table stored in AWS S3. You can obtain this URL from the AWS S3 console. The valid URL format is https://<bucket-name>.<s3_endpoint>/<prefix>. The NoSQL Database Migrator will look for json.gz files in the prefix during import.



You must export DynamoDB table as specified in <u>Exporting DynamoDB table data</u> to Amazon S3.

- Data Type: string
- Mandatory (Y/N): Y
- **Example:** https://my-bucket.s3.ap-south-1.amazonaws.com/AWSDynamoDB/01649660790057-14f642be

credentials

- **Purpose:** Specifies the absolute path to a file containing the AWS credentials. If not specified, it defaults to \$HOME/.aws/credentials. For more details on the credentials file, see <u>Configuration and credential file settings</u>.
- Data Type: string
- Mandatory (Y/N): N
- Example:

"credentials" : "/home/user/.aws/credentials"



"credentials" : "/home/user/security/credentials



(i) Note

The NoSQL Database Migrator does not log any of the credentials information. You must properly protect the credentials file from unauthorized access.

credentialsProfile

Purpose: Name of the profile in the AWS credentials file to be used to connect to AWS S3. User account credentials are referred to as a *profile*. If you do not specify this value, NoSQL Database Migrator uses the default profile. For more details on the credentials file, see Configuration and credential file settings.

Data Type: string

Mandatory (Y/N): N

Example:

```
"credentialsProfile" : "default"
"credentialsProfile" : "test"
```

ttlAttributeName

Purpose: Specifies the name of TTL attribute present in the exported DynamoDB table data. You include this parameter only if the DynamoDB table data has a TTL attribute and you want to set the TTL value on imported data while importing to NoSQL Database.



(i) Note

To import with the TTL metadata, you must set the includeTTL configuration parameter to true in the sink configuration template (nosqldb and nosqldb cloud).

- Data Type: string
- Mandatory (Y/N): N
- **Example:** "ttlAttributeName" : "ttl"

DynamoDB-Formatted JSON File

The configuration file format for DynamoDB-formatted JSON File as a source of NoSQL Database Migrator is shown below.

You can migrate a file or directory containing the DynamoDB exported JSON data from a file system by specifying the path in the source configuration template.

A sample DynamoDB-formatted JSON File is as follows:

```
{"Item":{"Id":{"N":"101"},"Phones":{"L":[{"L":[{"S":"555-222"},
{"S":"123-567"}]}]}, "PremierCustomer":{"BOOL":false}, "Address":{"M":{"Zip":
{"N":"570004"}, "Street":{"S":"21 main"}, "DoorNum":{"N":"201"}, "City":
{"S": "London"}}}, "FirstName": {"S": "Fred"}, "FavNumbers": {"NS":
["10"]}, "LastName": {"S": "Smith"}, "FavColors": {"SS": ["Red", "Green"]}, "Age":
{"N":"22"},"ttl": {"N": "1734616800"}}}
{"Item":{"Id":{"N":"102"},"Phones":{"L":[{"L":
```



```
[{"S":"222-222"}]}]}, "PremierCustomer":{"BOOL":false}, "Address":{"M":{"Zip":
{"N":"560014"}, "Street":{"S":"32 main"}, "DoorNum":{"N":"1024"}, "City":
{"S":"Wales"}}}, "FirstName":{"S":"John"}, "FavNumbers":{"NS":
["10"]}, "LastName":{"S":"White"}, "FavColors":{"SS":["Blue"]}, "Age":
{"N":"48"}, "ttl": {"N": "1734616800"}}}
```

You must copy the exported DynamoDB table data from AWS S3 storage to a local mounted file system.

The valid sink types for DynamoDB JSON file are nosqldb and nosqldb_cloud.

Source Configuration Template

```
"source" : {
  "type" : "file",
  "format" : "dynamodb_json",
  "ttlAttributeName" : <DynamoDB exported TTL attribute name>,
  "dataPath" : "<path/to/[file|dir]/containing/exported/DDB/tabledata>"
}
```

Source Parameters

Common Configuration Parameters

```
type
Use "type" : "file"

format
Use "format" : "dynamodb_json"
```

Unique Configuration Parameter

- dataPath
- ttlAttributeName

dataPath

- Purpose: Specifies the absolute path to a file or directory containing the exported DynamoDB table data. You must copy exported DynamoDB table data from AWS S3 to a local mounted file system. You must ensure that this data matches with the NoSQL table schema defined at the sink. If you specify a directory, the NoSQL Database Migrator identifies all the files with the .json.gz extension in that directory and the data subdirectory.
- Data Type: string
- Mandatory (Y/N): Y
- Example:
 - Specifying a file

```
"dataPath" : "/home/user/AWSDynamoDB/01639372501551-bb4dd8c3/data/zclclwucjy6v5mkefvckxzhfvq.json.gz"
```

Specifying a directory

"dataPath" : "/home/user/AWSDynamoDB/01639372501551-bb4dd8c3"



ttlAttributeName

Purpose: Specifies the name of TTL attribute present in the exported DynamoDB table data. You include this parameter only if the DynamoDB table data has a TTL attribute and you want to set the TTL value on imported data while importing to NoSQL Database.

(i) Note

To import with the TTL metadata, you must set the includeTTL configuration parameter to true in the sink configuration template (nosqldb and nosqldb_cloud).

- Data Type: string
- Mandatory (Y/N): N
- **Example:** "ttlAttributeName" : "ttl"

Oracle NoSQL Database

The configuration file format for Oracle NoSQL Database as a source of NoSQL Database Migrator is shown below.

You can migrate a table from Oracle NoSQL Database by specifying the table name in the source configuration template.

A sample Oracle NoSQL Database table is as follows:

```
{ "id":20, "firstName": "Jane", "lastName": "Smith", "otherNames":
[{"first":"Jane","last":"teacher"}],"age":25,"income":55000,"address":
{"city": "San Jose", "number": 201, "phones":
[{"area":608, "kind": "work", "number":6538955},
{ "area":931, "kind": "home", "number":9533341 },
{"area":931, "kind": "mobile", "number":9533382}], "state": "CA", "street": "Atlantic
Ave", "zip":95005}, "connections":[40,75,63], "expenses":null}
{"id":10,"firstName":"John","lastName":"Smith","otherNames":
[{"first":"Johny","last":"chef"}],"age":22,"income":45000,"address":
{"city": "Santa Cruz", "number": 101, "phones":
[{"area":408,"kind":"work","number":4538955},
{ "area":831, "kind": "home", "number":7533341 },
{"area":831, "kind": "mobile", "number":7533382}], "state": "CA", "street": "Pacific
Ave", "zip":95008}, "connections":[30,55,43], "expenses":null}
{"id":30,"firstName":"Adam","lastName":"Smith","otherNames":
[{"first":"Adam","last":"handyman"}],"age":45,"income":75000,"address":
{"city": "Houston", "number": 301, "phones":
[{"area":618,"kind":"work","number":6618955},
{ "area":951, "kind": "home", "number":9613341 },
{"area":981,"kind":"mobile","number":9613382}],"state":"TX","street":"Indian
Ave", "zip":95075}, "connections":[60,45,73], "expenses":null}
```

Source Configuration Template

```
"source" : {
  "type": "nosqldb",
  "storeName" : "<store name>",
  "helperHosts" : ["hostname1:port1", "hostname2:port2,..."],
```



```
"table" : "<fully qualified table name>",
  "includeTTL": <true|false>,
  "security" : "</path/to/store/security/file>",
  "requestTimeoutMs" : 5000
}
```

Source Parameters

Common Configuration Parameter

```
• type
   Use "type" : "nosqldb"
• security
   Example:
   "security" : "/home/user/client.credentials"
```

Example security file content for password file based authentication:

```
oracle.kv.password.noPrompt=true
oracle.kv.auth.username=admin
oracle.kv.auth.pwdfile.file=/home/nosql/login.passwd
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=/home/nosql/client.trust
oracle.kv.ssl.protocols=TLSv1.2
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

Example security file content for wallet based authentication:

```
oracle.kv.password.noPrompt=true
oracle.kv.auth.username=admin
oracle.kv.auth.wallet.dir=/home/nosql/login.wallet
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=/home/nosql/client.trust
oracle.kv.ssl.protocols=TLSv1.2
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

requestTimeoutMs

```
Example: "requestTimeoutMs" : 5000
```

Unique Configuration Parameters

- storeName
- <u>helperHosts</u>
- table
- includeTTL

storeName

- Purpose: Name of the Oracle NoSQL Database store.
- Data Type: string
- Mandatory (Y/N): Y
- Example: "storeName" : "kvstore"



helperHosts

- **Purpose:** A list of host and registry port pairs in the hostname:port format. Delimit each item in the list using a comma. You must specify at least one helper host.
- Data Type: array of strings
- Mandatory (Y/N): Y
- **Example:** "helperHosts" : ["localhost:5000","localhost:6000"]

table

• **Purpose:** Fully qualified table name from which to migrate the data.

```
Format: [namespace_name:]<table_name>
```

If the table is in the DEFAULT namespace, you can omit the namespace_name. The table must exist in the store.

- Data Type: string
- Mandatory (Y/N): Y
- Example:
 - With the DEFAULT namespace "table" : "mytable"
 - With a non-default namespace "table": "mynamespace:mytable"
 - To specify a child table "table": "mytable.child"

includeTTL

Purpose: Specifies whether or not to include TTL metadata for table rows when exporting
Oracle NoSQL Database tables. If set to true, the TTL data for rows also gets included in
the data provided by the source. TTL is present in the _metadata JSON object associated
with each row. The expiration time for each row gets exported as the number of
milliseconds since the UNIX epoch (Jan 1st, 1970).

If you do not specify this parameter, it defaults to false.

Only the rows having a positive expiration value for TTL get included as part of the exported rows. If a row does not expire, which means TTL=0, then its TTL metadata is not included explicitly. For example, if ROW1 expires at 2021-10-19 00:00:00 and ROW2 does not expire, the exported data looks like as follows:

```
//ROW1
{
    "id" : 1,
    "name" : "abc",
    "_metadata" : {
        "expiration" : 1634601600000
    }
}
//ROW2
{
    "id" : 2,
        "name" : "xyz"
}
```



Data Type: boolean

Mandatory (Y/N): N

Example: "includeTTL" : true

Oracle NoSQL Database Cloud Service

The configuration file format for Oracle NoSQL Database Cloud Service as a source of NoSQL Database Migrator is shown below.

You can migrate a table from Oracle NoSQL Database Cloud Service by specifying the name or OCID of the compartment in which the table resides in the source configuration template.

A sample Oracle NoSQL Database Cloud Service table is as follows:

```
{"id":20,"firstName":"Jane","lastName":"Smith","otherNames":
[{"first":"Jane","last":"teacher"}],"age":25,"income":55000,"address":
{"city":"San Jose","number":201,"phones":
[{"area":608, "kind": "work", "number":6538955},
{ "area":931, "kind": "home", "number":9533341 },
{"area":931, "kind": "mobile", "number":9533382}], "state": "CA", "street": "Atlantic
Ave", "zip":95005}, "connections":[40,75,63], "expenses":null}
{"id":10,"firstName":"John","lastName":"Smith","otherNames":
[{"first":"Johny","last":"chef"}],"age":22,"income":45000,"address":
{"city": "Santa Cruz", "number": 101, "phones":
[{"area":408, "kind": "work", "number":4538955},
{ "area":831, "kind": "home", "number ":7533341 },
{"area":831,"kind":"mobile","number":7533382}],"state":"CA","street":"Pacific
Ave", "zip":95008}, "connections":[30,55,43], "expenses":null}
{"id":30,"firstName":"Adam","lastName":"Smith","otherNames":
[{"first":"Adam","last":"handyman"}],"age":45,"income":75000,"address":
{"city": "Houston", "number": 301, "phones":
[{"area":618,"kind":"work","number":6618955},
{ "area":951, "kind": "home", "number":9613341},
{"area":981,"kind":"mobile","number":9613382}],"state":"TX","street":"Indian
Ave", "zip":95075}, "connections":[60,45,73], "expenses":null}
```

Source Configuration Template

```
"source" : {
  "type" : "nosqldb_cloud",
  "endpoint" : "<Oracle NoSQL Cloud Service endpoint URL or region ID>",
  "table" : "",
  "compartment" : "<OCI compartment name or id>",
  "credentials" : "<path/to/oci/credential/file>",
  "credentialsProfile" : "<profile name in oci config file>",
  "useInstancePrincipal" : <true|false>,
  "useDelegationToken" : <true|false>,
  "useSessionToken" : <true|false>,
  "useOKEWorkloadIdentity" : <true|false>,
  "readUnitsPercent" : ,
  "includeTTL": <true|false>,
  "requestTimeoutMs" : <timeout in milli seconds>
```



Source Parameters

Common Configuration Parameters

```
type
Use "type" : "nosqldb_cloud"
```

endpoint

Example:

- Region ID: "endpoint" : "us-ashburn-1"
- URL format: "endpoint" : "https://objectstorage.us-ashburn-1.oraclecloud.com"
- credentials

Example:

- 1. "credentials" : "/home/user/.oci/config"
- 2. "credentials" : "/home/user/security/config"
- credentialsProfile

Example:

- 1. "credentialsProfile" : "DEFAULT"
- 2. "credentialsProfile" : "ADMIN_USER"
- <u>useInstancePrincipal</u>

Example: "useInstancePrincipal" : true

• <u>useDelegationToken</u>

Example: "useDelegationToken" : true

Note

The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

useSessionToken

Example: "useSessionToken" : true

requestTimeoutMs

Example: "requestTimeoutMs" : 5000

Unique Configuration Parameters

- table
- compartment
- readUnitsPercent
- includeTTL

table

- Purpose: Name of the table from which to migrate the data.
- Data Type: string



- Mandatory (Y/N): Y
- Example:
 - To specify a table "table" : "myTable"
 - To specify a child table "table" : "mytable.child"

compartment

Purpose: Specifies the name or OCID of the compartment in which the table resides.

If you do not provide any value, it defaults to the *root* compartment.

You can find your compartment's OCID from the Compartment Explorer window under Governance in the OCI Cloud Console.

- Data Type: string
- **Mandatory (Y/N):** Y, if the table is not in the root compartment of the tenancy OR when the useInstancePrincipal parameter is set to true.

① Note

If the useInstancePrincipal parameter is set to true, the compartment must specify the compartment OCID and not the name.

Example:

Compartment name

```
"compartment" : "mycompartment"
```

Compartment name qualified with its parent compartment

```
"compartment" : "parent.childcompartment"
```

No value provided. Defaults to the root compartment.

```
"compartment": ""
```

Compartment OCID

```
"compartment" : "ocid1.tenancy.oc1...4ksd"
```

readUnitsPercent

Purpose: Percentage of table read units to be used while migrating the NoSQL table.

The default value is 90. The valid range is any integer between 1 to 100. The amount of time required to migrate data is directly proportional to this attribute. It is better to increase the read throughput of the table for the migration activity. You can reduce the read throughput after the migration process completes.

To learn the daily limits on throughput changes, see Cloud Limits in the *Oracle NoSQL Database Cloud Service* document.

See <u>Troubleshooting the Oracle NoSQL Database Migrator</u> to learn how to use this attribute to improve the data migration speed.

- Data Type: integer
- Mandatory (Y/N): N
- **Example:** "readUnitsPercent" : 90



includeTTL

Purpose: Specifies whether or not to include TTL metadata for table rows when exporting
Oracle NoSQL Database tables. If set to true, the TTL data for rows also gets included in
the data provided by the source. TTL is present in the _metadata JSON object associated
with each row. The expiration time for each row gets exported as the number of
milliseconds since the UNIX epoch (Jan 1st, 1970).

If you do not specify this parameter, it defaults to false.

Only the rows having a positive expiration value for TTL get included as part of the exported rows. If a row does not expire, which means TTL=0, then its TTL metadata is not included explicitly. For example, if ROW1 expires at 2021-10-19 00:00:00 and ROW2 does not expire, the exported data looks like as follows:

```
//ROW1
{
    "id" : 1,
    "name" : "abc",
    "_metadata" : {
        "expiration" : 1634601600000
    }
}

//ROW2
{
    "id" : 2,
    "name" : "xyz"
}
```

- Data Type: boolean
- Mandatory (Y/N): N
- Example: "includeTTL" : true

CSV File Source

The configuration file format for the CSV file as a source of NoSQL Database Migrator is shown below. The CSV file must conform to the RFC4180 format.

You can migrate a CSV file or a directory containing the CSV data by specifying the file name or directory in the source configuration template.

A sample CSV file is as follows:

```
1, "Computer Science", "San Francisco", "2500"
2, "Bio-Technology", "Los Angeles", "1200"
3, "Journalism", "Las Vegas", "1500"
4, "Telecommunication", "San Francisco", "2500"
```

Source Configuration Template

```
"source" : {
  "type" : "file",
  "format" : "csv",
  "dataPath": "</path/to/a/csv/[file|dir]>",
```



```
"hasHeader" : <true | false>,
  "columns" : ["column1", "column2", ....],
  "csvOptions": {
    "encoding": "<character set encoding>",
        "trim": "<true | false>"
}
```

Source Parameters

Common Configuration Parameters

```
    type
        Use "type" : "file"
    format
        Use "format" : "csv"
```

Unique Configuration Parameters

- dataPath
- hasHeader
- columns
- <u>csvOptions</u>
- <u>csvOptions.encoding</u>
- csvOptions.trim

datapath

• **Purpose:** Specifies the absolute path to a file or directory containing the CSV data for migration. If you specify a directory, NoSQL Database Migrator imports all the files with the .csv or .csv extension in that directory. All the CSV files are copied into a single table, but not in any particular order.

CSV files must conform to the RFC4180 standard. You must ensure that the data in each CSV file matches with the NoSQL Database table schema defined in the sink table. Subdirectories are not supported.

- Data Type: string
- Mandatory (Y/N): Y
- Example:
 - Specifying a CSV file

```
"dataPath" : "/home/user/sample.csv"
```

Specifying a directory

"dataPath" : "/home/user"



Note

The CSV files must contain only scalar values. Importing CSV files containing complex types such as MAP, RECORD, ARRAY, and JSON is not supported. The NoSQL Database Migrator tool does not check for the correctness of the data in the input CSV file. The NoSQL Database Migrator tool supports the importing of CSV data that conforms to the RFC4180 format. CSV files containing data that does not conform to the RFC4180 standard may not get copied correctly or may result in an error. If the input data is corrupted, the NoSQL Database Migrator tool will not parse the CSV records. If any errors are encountered during migration, the NoSQL Database Migrator tool logs the information about the failed input records for debugging and informative purposes. For more details, see *Logging Migrator Progress* in Using Oracle NoSQL Data Migrator.

hasHeader

- Purpose: Specifies if the CSV file has a header or not. If this is set to true, the first line is ignored. If it is set to false, the first line is considered a CSV record. The default value is false.
- Data Type: BooleanMandatory (Y/N): N
- Example: "hasHeader" : "false"

columns

• **Purpose:** Specifies the list of NoSQL Database table column names. The order of the column names indicates the mapping of the CSV file fields with corresponding NoSQL Database table columns. If the order of the input CSV file columns does not match the existing or newly created NoSQL Database table columns, you can map the ordering using this parameter. Also, when importing into a table that has an Identity Column, you can skip the Identity column name in the columns parameter.

Note

- If the NoSQL Database table has additional columns that are not available in the CSV file, the values of the missing columns are updated with the default value as defined in the NoSQL Database table. If a default value is not provided, a Null value is inserted during migration. For more information on default values, see Data Type Definitions section in the SQL Reference Guide.
- If the CSV file has additional columns that are not defined in the NoSQL Database table, the additional column information is ignored.
- If any value in the CSV record is empty, it is set to the default value of the corresponding columns in the NoSQL Database table. If a default value is not provided, a Null value is inserted during migration.
- Data Type: Array of Strings
- Mandatory (Y/N): N
- **Example:** "columns" : ["table_column_1", "table_column_2"]



csvOptions

- **Purpose:** Specifies the formatting options for a CSV file. Provide the character set encoding format of the CSV file and choose whether or not to trim the blank spaces.
- Data Type: Object
- Mandatory (Y/N): N

csvOptions.encoding

- **Purpose:** Specifies the character set to decode the CSV file. The default value is UTF-8. The supported character sets are US-ASCII, ISO-8859-1, UTF-8, and UTF-16.
- Data Type: String
- Mandatory (Y/N): N
- Example: "encoding" : "UTF-8"

csvOptions.trim

- Purpose: Specifies if the leading and trailing blanks of a CSV field value must be trimmed.
 The default value is false.
- Data Type: Boolean
- Mandatory (Y/N): N
- Example: "trim" : "true"

CSV file in OCI Object Storage Bucket

The configuration file format for the CSV file in OCI Object Storage bucket as a source of NoSQL Database Migrator is shown below. The CSV file must conform to the RFC4180 format.

You can migrate a CSV file in the OCI Object Storage bucket by specifying the name of the bucket in the source configuration template.

A sample CSV file in the OCI Object Storage bucket is as follows:

```
1,"Computer Science","San Francisco","2500"
2,"Bio-Technology","Los Angeles","1200"
3,"Journalism","Las Vegas","1500"
4,"Telecommunication","San Francisco","2500"
```

Note

The valid sink types for OCI Object Storage source type are nosqldb and $nosqldb_cloud$.

Source Configuration Template

```
"source" : {
  "type" : "object_storage_oci",
  "format" : "csv",
  "endpoint" : "<OCI Object Storage service endpoint URL or region ID>",
  "namespace" : "<OCI Object Storage namespace>",
```



```
"bucket" : "<bucket name>",
"prefix" : "<object prefix>",
"credentials" : "</path/to/oci/config/file>",
"credentialsProfile" : "profile name in oci config file>",
"useInstancePrincipal" : <true|false>,
"useDelegationToken" : <true|false>,
"useSessionToken" : <true|false>,
"useOKEWorkloadIdentity" : <true|false>,
"hasHeader" : <true | false>,
"columns" : ["column1", "column2", ....],
"csvOptions" : {
    "encoding" : "<character set encoding>",
    "trim" : <true | false>
}
}
```

Source Parameters

Common Configuration Parameters

Note

- The NoSQL Database Migrator imports all the files with the .csv or .Csv extension object-wise and copies them into a single table in the same order.
- The CSV files must contain only scalar values. Importing CSV files containing complex types such as MAP, RECORD, ARRAY, and JSON is not supported. The NoSQL Database Migrator tool does not check for the correctness of the data in the input CSV file. The NoSQL Database Migrator tool supports the importing of CSV data that conforms to the RFC4180 format. CSV files containing data that does not conform to the RFC4180 standard may not get copied correctly or may result in an error. If the input data is corrupted, the NoSQL Database Migrator tool will not parse the CSV records. If any errors are encountered during migration, the NoSQL Database Migrator tool logs the information about the failed input records for debugging and informative purposes. For more details, see Logging Migrator Progress in Using Oracle NoSQL Data Migrator.



prefix

Example:

- 1. "prefix" : "my_table/Data/000000.csv" (migrates only 000000.csv)
- 2. "prefix": "my_table/Data" (migrates all the objects with prefix my_table/Data)
- credentials

Example:

- 1. "credentials" : "/home/user/.oci/config"
- 2. "credentials" : "/home/user/security/config"
- credentialsProfile

Example:

- 1. "credentialsProfile" : "DEFAULT"
- 2. "credentialsProfile" : "ADMIN_USER"
- useInstancePrincipal

Example: "useInstancePrincipal" : true

useDelegationToken

Example: "useDelegationToken" : true



The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

<u>useSessionToken</u>

Example: "useSessionToken" : true

Unique Configuration Parameters

- <u>hasHeader</u>
- <u>columns</u>
- csvOptions
- csvOptions.encoding
- csvOptions.trim

hasHeader

- **Purpose:** Specifies if the CSV file has a header or not. If this is set to true, the first line is ignored. If it is set to false, the first line is considered a CSV record. The default value is false.
- Data Type: Boolean
- Mandatory (Y/N): N
- Example: "hasHeader" : "false"



columns

Purpose: Specifies the list of NoSQL Database table column names. The order of the column names indicates the mapping of the CSV file fields with corresponding NoSQL Database table columns. If the order of the input CSV file columns does not match the existing or newly created NoSQL Database table columns, you can map the ordering using this parameter. Also, when importing into a table that has an Identity Column, you can skip the Identity column name in the columns parameter.

Note

- If the NoSQL Database table has additional columns that are not available in the CSV file, the values of the missing columns are updated with the default value as defined in the NoSQL Database table. If a default value is not provided, a Null value is inserted during migration. For more information on default values, see Data Type Definitions section in the SQL Reference Guide.
- If the CSV file has additional columns that are not defined in the NoSQL Database table, the additional column information is ignored.
- If any value in the CSV record is empty, it is set to the default value of the corresponding columns in the NoSQL Database table. If a default value is not provided, a Null value is inserted during migration.
- Data Type: Array of Strings
- Mandatory (Y/N): N
- **Example:** "columns" : ["table_column_1", "table_column_2"]

csvOptions

- **Purpose:** Specifies the formatting options for a CSV file. Provide the character set encoding format of the CSV file and choose whether or not to trim the blank spaces.
- Data Type: Object
- Mandatory (Y/N): N

csvOptions.encoding

- **Purpose:** Specifies the character set to decode the CSV file. The default value is UTF-8. The supported character sets are US-ASCII, ISO-8859-1, UTF-8, and UTF-16.
- Data Type: String
- Mandatory (Y/N): N
- Example: "encoding" : "UTF-8"

csvOptions.trim

- Purpose: Specifies if the leading and trailing blanks of a CSV field value must be trimmed.
 The default value is false.
- Data Type: Boolean
- Mandatory (Y/N): N
- Example: "trim" : "true"



Sink Configuration Templates

Learn about the sink configuration file formats for each valid sink and the purpose of each configuration parameter.

For the configuration file template, see **Configuration File** in Terminology used with Oracle NoSQL Database Migrator.

For details on valid source formats for each of the sinks, see Source Configuration Templates.

Topics

The following topics describe the sink configuration templates referred by Oracle NoSQL Database Migrator to copy the data from a valid source to the given sink.

- JSON File Sink Specified JSON file.
- Parquet File

Parguet file in the specified directory.

- JSON File in OCI Object Storage Bucket
 JSON file in the specified OCI Object Storage bucket.
- Parquet File in OCI Object Storage Bucket
 Parquet file in the specified OCI Object Storage bucket.
- Oracle NoSQL Database
 Specified table in Oracle NoSQL Database.
- Oracle NoSQL Database Cloud Service
 Specified table in Oracle NoSQL Database Cloud Service.

JSON File Sink

The configuration file format for JSON File as a sink of NoSQL Database Migrator is shown below.

Sink Configuration Template

```
"sink" : {
  "type" : "file",
  "format" : "json",
  "dataPath": "</path/to/a/directory>",
  "schemaPath" : "<path/to/a/file>",
  "pretty" : <true|false>,
  "useMultiFiles" : <true|false>,
  "chunkSize" : <size in MB>
}
```

Sink Parameters

Common Configuration Parameters

```
type
Use "type" : "file"format
Use "format" : "json"
```



chunkSize

Example: "chunkSize" : 40



(i) Note

This parameter is applicable ONLY when the useMultiFiles parameter is set to

Unique Configuration Parameters

- dataPath
- schemaPath
- pretty
- useMultiFiles

dataPath

Purpose: Specifies the path to a directory where NoSQL Database Migrator copies the source data in the JSON format.

NoSQL Database Migrator creates JSON files in the specified directory. If the files exist, NoSQL Database Migrator overwrites their content with source data.

Ensure that the directory already exists and has read and write permissions.

- **Data Type:** string
- Mandatory (Y/N): Y
- **Example:** "dataPath" : "/home/user/data"

After successful migration, the directory specified in the dataPath parameter will include exported files as shown in the following sample:

```
--<Table_name>_1_5.json
|--<Table_name>_6_10.json
```

schemaPath

Purpose: Specifies the absolute path to a file to write table schema information provided by the source.

If this value is not defined, the source schema information will not be migrated to the sink. If this value is specified, the migrator utility writes the schema of the source table into the file specified here.

The schema information is written as one DDL command per line in this file. If the file does not exist in the specified data path, NoSQL Database Migrator creates it. If it exists already, NoSQL Database Migrator will overwrite its contents with the source data. You must ensure that the parent directory in the data path is valid for the specified file.

- Data Type: string
- Mandatory (Y/N): N
- **Example:** "schemaPath" : "/home/user/schema_file"



pretty

Purpose: Specifies whether or not to beautify the JSON output to increase readability.
 If not specified, it defaults to false.

Data Type: boolean
 Mandatory (Y/N): N
 Example: "pretty" : true

useMultiFiles

• **Purpose:** Specifies whether or not to further split the exported files (created under the directory specified in the dataPath parameter) into multiple sub-files of a specific size while migrating NoSQL Database table data to a directory. The useMultiFiles parameter defaults to true.

NoSQL Database Migrator splits the NoSQL Database table data into multiple files while exporting data. If useMultiFiles parameter is set to true, each exported file is further split into sub-files of size specified in the chunkSize parameter.

Example: After successful migration, the directory specified in the dataPath parameter will include exported files as shown in the following sample:

```
|--<Table_name>_1_5_0.json
|--<Table_name>_1_5_1.json
|--<Table_name>_6_10_0.json
|--<Table_name>_6_10_1.json
|--<Table_name>_6_10_2.json
```

- Data Type: boolean
- Mandatory (Y/N): N
- Example: "useMultiFiles" : true

Parquet File

The configuration file format for Parquet File as a sink of NoSQL Database Migrator is shown below.

Sink Configuration Template

```
"sink" : {
  "type" : "file",
  "format" : "parquet",
  "dataPath": "</path/to/a/dir>",
  "chunkSize" : <size in MB>,
  "compression": "<SNAPPY|GZIP|NONE>",
  "parquetOptions": {
      "useLogicalJson": <true|false>,
      "useLogicalEnum": <true|false>,
      "useLogicalUUID": <true|false>,
      "truncateDoubleSpecials": <true|false>
```



```
}
}
```

Sink Parameters

Common Configuration Parameters

```
type
Use "type" : "file"

format
Use "format" : "parquet"

chunkSize
Example: "chunkSize" : 40
```

Unique Configuration Parameters

- dataPath
- compression
- parquetOptions
- parquetOptions.useLogicalJson
- parquetOptions.useLogicalEnum
- parquetOptions.useLogicalUUID
- parquetOptions.truncateDoubleSpecials

dataPath

- **Purpose:** Specifies the path to a directory for storing the migrated NoSQL table data. Ensure that the directory already exists and has read and write permissions.
- Data Type: string
- Mandatory (Y/N): Y
- **Example:** "dataPath" : "/home/user/migrator/my_table"

compression

Purpose: Specifies the compression type to use to compress the Parquet data. Valid values are SNAPPY, GZIP, and NONE.

If not specified, it defaults to SNAPPY.

- Data Type: string
- Mandatory (Y/N): N
- **Example:** "compression" : "GZIP"

parquetOptions

 Purpose: Specifies the options to select Parquet logical types for NoSQL ENUM, JSON, and UUID columns.

If you do not specify this parameter, the NoSQL Database Migrator writes the data of ENUM, JSON, and UUID columns as String.

- Data Type: object
- Mandatory (Y/N): N



parquetOptions.useLogicalJson

 Purpose: Specifies whether or not to write NoSQL JSON column data as Parquet logical JSON type. For more information, see Parquet Logical Type Definitions.

If not specified or set to false, NoSQL Database Migrator writes the NoSQL JSON column data as String.

- Data Type: booleanMandatory (Y/N): N
- Example: "useLogicalJson" : true

parquetOptions.useLogicalEnum

 Purpose: Specifies whether or not to write NoSQL ENUM column data as Parquet logical ENUM type. For more information, see <u>Parquet Logical Type Definitions</u>.

If not specified or set to false, NoSQL Database Migrator writes the NoSQL ENUM column data as String.

- Data Type: boolean
- Mandatory (Y/N): N
- **Example:** "useLogicalEnum" : true

parquetOptions.useLogicalUUID

 Purpose: Specifies whether or not to write NoSQL UUID column data as Parquet logical UUID type. For more information, see Parquet Logical Type Definitions.

If not specified or set to false, NoSQL Database Migrator writes the NoSQL UUID column data as String.

- Data Type: boolean
- Mandatory (Y/N): N
- Example: "useLogicalUUID" : true

parquetOptions.truncateDoubleSpecials

Purpose: Specifies whether or not to truncate the double +Infinity, -Infinity, and NaN values.

By default, it is set to false. If set to true,

- Positive Infinity is truncated to Double.MAX VALUE.
- NEGATIVE_INFINITY is truncated to -Double.MAX_VALUE.
- NaN is truncated to 9.999999999999990E307.
- Data Type: boolean
- Mandatory (Y/N): N
- **Example:** "truncateDoubleSpecials" : true



JSON File in OCI Object Storage Bucket

The configuration file format for JSON file in OCI Object Storage bucket as a sink of NoSQL Database Migrator is shown below.

(i) Note

The valid source types for OCI Object Storage as the sink are nosqldb and nosqldb_cloud.

Sink Configuration Template

```
"sink" : {
  "type" : "object_storage_oci",
  "format" : "json",
  "endpoint" : "<OCI Object Storage service endpoint URL or region ID>",
  "namespace" : "<OCI Object Storage namespace>",
  "bucket" : "<bucket name>",
  "prefix" : "<object prefix>",
  "chunkSize" : <size in MB>,
  "pretty" : <true|false>,
  "credentials" : "</path/to/oci/config/file>",
  "credentialsProfile" : "<profile name in oci config file>",
  "useInstancePrincipal" : <true|false>,
  "useDelegationToken" : <true|false>,
  "useSessionToken" : <true|false>,
  "useOKEWorkloadIdentity" : <true|false>,
  "useOKEWorkloadIdentity" : <true|false>,
}
```

Sink Parameters

Common Configuration Parameters

```
type
Use "type" : "object_storage_oci"

format
Use "format" : "json"

endpoint
Example:

- Region ID: "endpoint" : "us-ashburn-1"

- URL format: "endpoint" : "https://objectstorage.us-ashburn-1.oraclecloud.com"

namespace
Example: "namespace" : "my-namespace"

bucket
Example: "bucket" : "my-bucket"

prefix
```



Schema is migrated to the <prefix>/Schema.ddl file and source data is migrated to the <prefix>/Data/<chunk>. json files, where chunk=000000.json, 000001.json, and so forth.

Example:

```
    "prefix" : "my_export"
    "prefix" : "my_export/2021-04-05/"
```

chunkSize

Example: "chunkSize" : 40

credentials

Example:

```
    "credentials": "/home/user/.oci/config"
    "credentials": "/home/user/security/config"
```

credentialsProfile

Example:

```
    "credentialsProfile" : "DEFAULT"
    "credentialsProfile" : "ADMIN_USER"
```

• <u>useInstancePrincipal</u>

Example: "useInstancePrincipal" : true

useDelegationToken

Example: "useDelegationToken" : true

(i) Note

The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

useSessionToken

Example: "useSessionToken" : true

Unique Configuration Parameter

pretty

Purpose: Specifies whether or not to beautify the JSON output to increase readability.
 If not specified, it defaults to false.

Data Type: boolean

Mandatory (Y/N): N

• **Example:** "pretty" : true



Parquet File in OCI Object Storage Bucket

The configuration file format for Parquet file in OCI Object Storage bucket as a sink of NoSQL Database Migrator is shown below.

Note

The valid source types for OCI Object Storage source type are nosqldb and nosqldb_cloud.

Sink Configuration Template

```
"sink" : {
  "type" : "object storage oci",
  "format" : "parquet",
  "endpoint" : "<OCI Object Storage service endpoint URL or region ID>",
  "namespace" : "<OCI Object Storage namespace>",
  "bucket" : "<bucket name>",
  "prefix" : "<object prefix>",
  "chunkSize" : <size in MB>,
  "compression": "<SNAPPY GZIP NONE>",
  "parquetOptions": {
    "useLogicalJson": <true | false>,
    "useLogicalEnum": <true | false>,
    "useLogicalUUID": <true false>,
    "truncateDoubleSpecials": <true | false>
  "credentials" : "</path/to/oci/config/file>",
  "credentialsProfile" : "<profile name in oci config file>",
  "useInstancePrincipal" : <true | false>,
  "useDelegationToken" : <true | false>,
  "useSessionToken" : <true | false>,
  "useOKEWorkloadIdentity" : <true | false>
```

Sink Parameters

Common Configuration Parameters

```
type
Use "type" : "object_storage_oci"

format
Use "format" : "parquet"

endpoint
Example:

Region ID: "endpoint" : "us-ashburn-1"

URL format: "endpoint" : "https://objectstorage.us-ashburn-1.oraclecloud.com"
```

Example: "namespace" : "my-namespace"



bucket

Example: "bucket" : "my-bucket"

prefix

Source data is migrated to the <prefix>/Data/<chunk>.parquet files, where chunk=000000.parquet, 000001.parquet, and so forth.

Example:

- 1. "prefix" : "my_export"
- 2. "prefix" : "my export/2021-04-05/"
- chunkSize

Example: "chunkSize" : 40

credentials

Example:

- 1. "credentials" : "/home/user/.oci/config"
- 2. "credentials" : "/home/user/security/config"
- credentialsProfile

Example:

- 1. "credentialsProfile" : "DEFAULT"
- 2. "credentialsProfile" : "ADMIN_USER"
- useInstancePrincipal

Example: "useInstancePrincipal" : true

<u>useDelegationToken</u>

Example: "useDelegationToken" : true



The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

useSessionToken

Example: "useSessionToken" : true

Unique Configuration Parameter

- compression
- parquetOptions
- parquetOptions.useLogicalJson
- parquetOptions.useLogicalEnum
- parquetOptions.useLogicalUUID
- parquetOptions.truncateDoubleSpecials

compression

• **Purpose:** Specifies the compression type to use to compress the Parquet data. Valid values are SNAPPY, GZIP, and NONE.



If not specified, it defaults to SNAPPY.

Data Type: stringMandatory (Y/N): N

Example: "compression" : "GZIP"

parquetOptions

 Purpose: Specifies the options to select Parquet logical types for NoSQL ENUM, JSON, and UUID columns.

If you do not specify this parameter, the NoSQL Database Migrator writes the data of ENUM, JSON, and UUID columns as String.

Data Type: object

Mandatory (Y/N): N

parquetOptions.useLogicalJson

• **Purpose:** Specifies whether or not to write NoSQL JSON column data as Parquet logical JSON type. For more information, see Parquet Logical Type Definitions.

If not specified or set to false, NoSQL Database Migrator writes the NoSQL JSON column data as String.

Data Type: booleanMandatory (Y/N): N

• **Example:** "useLogicalJson" : true

parquetOptions.useLogicalEnum

 Purpose: Specifies whether or not to write NoSQL ENUM column data as Parquet logical ENUM type. For more information, see Parquet Logical Type Definitions.

If not specified or set to false, NoSQL Database Migrator writes the NoSQL ENUM column data as String.

• Data Type: boolean

Mandatory (Y/N): N

Example: "useLogicalEnum" : true

parquetOptions.useLogicalUUID

 Purpose: Specifies whether or not to write NoSQL UUID column data as Parquet logical UUID type. For more information, see <u>Parquet Logical Type Definitions</u>.

If not specified or set to false, NoSQL Database Migrator writes the NoSQL UUID column data as String.

Data Type: boolean

Mandatory (Y/N): N

Example: "useLogicalUUID" : true

parquetOptions.truncateDoubleSpecials

Purpose: Specifies whether or not to truncate the double +Infinity, -Infinity, and NaN values.



By default, it is set to false. If set to true,

- Positive_Infinity is truncated to Double.MAX_VALUE.
- NEGATIVE_INFINITY is truncated to -Double.MAX_VALUE.
- NaN is truncated to 9.999999999999990E307.
- Data Type: boolean
- Mandatory (Y/N): N
- **Example:** "truncateDoubleSpecials" : true

Oracle NoSQL Database

The configuration file format for Oracle NoSQL Database as a sink of NoSQL Database Migrator is shown below.

Sink Configuration Template

```
"sink" : {
  "type": "nosqldb",
  "storeName" : "<store name>",
 "helperHosts" : ["hostname1:port1", "hostname2:port2,..."],
  "security" : "</path/to/store/credentials/file>",
  "table" : "<fully qualified table name>",
 "includeTTL": <true false>,
  "ttlRelativeDate": "<date-to-use in UTC>",
  "schemaInfo" : {
    "schemaPath" : "</path/to/a/schema/file>",
    "defaultSchema" : <true | false>,
    "useSourceSchema" : <true | false>,
    "DDBPartitionKey" : <"name:type">,
    "DDBSortKey" : "<name:type>"
 },
 "overwrite" : <true | false>,
  "requestTimeoutMs" : <timeout in milli seconds>
```

Sink Parameters

Common Configuration Parameter

oracle.kv.transport=ssl

```
Use "type" : "nosqldb"

security
Example:

"security" : "/home/user/client.credentials"

Example security file content for password file based authentication:

oracle.kv.password.noPrompt=true
oracle.kv.auth.username=admin
oracle.kv.auth.pwdfile.file=/home/nosql/login.passwd
```

oracle.kv.ssl.trustStore=/home/nosql/client.trust



```
oracle.kv.ssl.protocols=TLSv1.2
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

Example security file content for wallet based authentication:

```
oracle.kv.password.noPrompt=true
oracle.kv.auth.username=admin
oracle.kv.auth.wallet.dir=/home/nosql/login.wallet
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=/home/nosql/client.trust
oracle.kv.ssl.protocols=TLSv1.2
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

requestTimeoutMs

Example: "requestTimeoutMs" : 5000

Unique Configuration Parameter

- storeName
- helperHosts
- <u>table</u>
- includeTTL
- ttlRelativeDate
- schemalnfo
- schemaInfo.schemaPath
- schemaInfo.defaultSchema
- schemaInfo.useSourceSchema
- schemaInfo.DDBPartitionKey
- schemalnfo.DDBSortKey
- overwrite

storeName

- **Purpose:** Name of the Oracle NoSQL Database store.
- Data Type: string
- Mandatory (Y/N): Y
- Example: "storeName" : "kvstore"

helperHosts

- **Purpose:** A list of host and registry port pairs in the hostname:port format. Delimit each item in the list using a comma. You must specify at least one helper host.
- Data Type: array of strings
- Mandatory (Y/N): Y
- Example: "helperHosts" : ["localhost:5000","localhost:6000"]

table

Purpose: Specifies the table name to store the migrated data.



Format: [namespace_name:]<table_name>

If the table is in the DEFAULT namespace, you can omit the <code>namespace_name</code>. The table must exist in the store during the migration, and its schema must match with the source data.

If the table is not available in the sink, you can use the schemaInfo parameter to instruct the NoSQL Database Migrator to create the table in the sink.

- Data Type: string
- Mandatory (Y/N): Y
- Example:
 - With the DEFAULT namespace "table" : "mytable"
 - With a non-default namespace "table" : "mynamespace:mytable"
 - To specify a child table "table": "mytable.child"



You can migrate the child tables from a valid data source to Oracle NoSQL Database. The NoSQL Database Migrator copies only a single table in each execution. Ensure that the parent table is migrated before the child table.

includeTTL

 Purpose: Specifies whether or not to include TTL metadata for table rows provided by the source when importing Oracle NoSQL Database tables.

If you do not specify this parameter, it defaults to false. In that case, the NoSQL Database Migrator does not include TTL metadata for table rows provided by the source when importing Oracle NoSQL Database tables.

If set to true, the NoSQL Database Migrator tool performs the following checks on the TTL metadata when importing a table row:

- If you import a row that does not have _metadata definition, the NoSQL Database
 Migrator tool sets the TTL to 0, which means the row never expires.
- If you import a row that has _metadata definition, the NoSQL Database Migrator tool compares the TTL value against a Reference Time when a row gets imported. If the row has already expired relative to the Reference Time, then it is skipped. If the row has not expired, then it is imported along with the TTL value. By default, the Reference Time of import operation is the current time in milliseconds, obtained from System.currentTimeMillis(), of the machine where the NoSQL Database Migrator tool is running. But you can also set a custom Reference Time using the ttlRelativeDate configuration parameter if you want to extend the expiration time and import rows that would otherwise expire immediately.

The formula to calculate the expiration time of a row is as follows:

expiration = (TTL value of source row in milliseconds - Reference Time
in milliseconds)
if (expiration <= 0) then it indicates that row has expired.</pre>



Note

Since Oracle NoSQL TTL boundaries are in hours and days, in some cases, the TTL of the imported row might get adjusted to the nearest hour or day. For example, consider a row that has expiration value of 1629709200000 (2021-08-23 09:00:00) and Reference Time value is 1629707962582 (2021-08-23 08:39:22). Here, even though the row is not expired relative to the Reference Time when this data gets imported, the new TTL for the row is 1629712800000 (2021-08-23 10:00:00).

Data Type: booleanMandatory (Y/N): N

• Example: "includeTTL" : true

ttlRelativeDate

 Purpose: Specify a UTC date in the YYYY-MM-DD hh:mm:ss format used to set the TTL expiry of table rows during importing into the Oracle NoSQL Database.

If a table row in the data you are exporting has expired, you can set the ttlRelativeDate parameter to a date before the expiration time of the table row in the exported data.

If you do not specify this parameter, it defaults to the current time in milliseconds, obtained from System.currentTimeMillis(), of the machine where the NoSQL Database Migrator tool is running.

Data Type: date

Mandatory (Y/N): N

Example: "ttlRelativeDate" : "2021-01-03 04:31:17"

Let us consider a scenario where table rows expire after seven days from 1-Jan-2021. After exporting this table, on 7-Jan-2021, you run into an issue with your table and decide to import the data. The table rows are going to expire in one day (data expiration date minus the default value of ttlRelativedate configuration parameter, which is the current date). But if you want to extend the expiration date of table rows to five days instead of one day, use the ttlRelativeDate parameter and choose an earlier date. Therefore, in this scenario if you want to extend expiration time of the table rows by five days, set the value of ttlRelativeDate configuration parameters to 3-Jan-2021, which is used as Reference Time when table rows get imported.

schemainfo

- **Purpose:** Specifies the schema for the data being migrated. If this is not specified, the NoSQL Database Migrator assumes that the table already exists in the sink's store.
- Data Type: Object
- Mandatory (Y/N): N

schemalnfo.schemaPath

Purpose: Specifies the absolute path to a file containing DDL statements for the NoSQL table.

The NoSQL Database Migrator executes the DDL commands listed in this file before migrating the data.



The NoSQL Database Migrator does not support more than one DDL statement per line in the schemaPath file.

- Data Type: string
- Mandatory (Y/N): N

Note

defaultSchema and schemaPath are mutually exclusive.

Example: "schemaPath" : "/home/user/schema_file"

schemaInfo.defaultSchema

- Purpose: Setting this parameter to true instructs the NoSQL Database Migrator to create a
 table with default schema. The default schema is defined by the migrator itself. For more
 information about default schema definitions, see Default Schema in Using Oracle NoSQL
 Data Migrator.
- Data Type: boolean
- Mandatory (Y/N): N

(i) Note

defaultSchema and schemaPath are mutually exclusive.

schemaInfo.useSourceSchema

- Purpose: Specifies whether or not the sink uses the table schema definition provided by the source when migrating NoSQL tables.
- Data Type: boolean
- Mandatory (Y/N): N

Note

defaultSchema, schemaPath, and useSourceSchema parameters are mutually exclusive. Specify only one of these parameters.

- Example:
 - With Default Schema:

```
"schemaInfo" : {
   "defaultSchema" : true
}
```

With a pre-defined schema:

```
"schemaInfo" : {
  "schemaPath" : "<complete/path/to/the/schema/definition/file>"
}
```



With source schema:

```
"schemaInfo" : {
   "useSourceSchema" : true
}
```

schemaInfo.DDBPartitionKey

- **Purpose:** Specifies the DynamoDB partition key and the corresponding Oracle NoSQL Database type to be used in the sink Oracle NoSQL Database table. This key will be used as a NoSQL DB table shard key. This is applicable only when defaultSchema is set to true and the source format is dynamodb_json. See Mapping of DynamoDB types to Oracle NoSQL types for more details.
- Mandatory (Y/N): Y, if defaultSchema is true and the source is dynamodb_json.
- **Example:** "DDBPartitionKey" : "PersonID:INTEGER"

Note

If the partition key contains dash(-) or dot(.), Migrator will replace it with underscore(_) as NoSQL column name does not support dot and dash.

schemalnfo.DDBSortKey

- **Purpose:** Specifies the DynamoDB sort key and its corresponding Oracle NoSQL Database type to be used in the target Oracle NoSQL Database table. If the importing DynamoDB table does not have a sort key, this attribute must not be set. This key will be used as a non-shard portion of the primary key in the NoSQL DB table. This is applicable only when defaultSchema is set to true and the source is dynamodb_json. See Mapping of DynamoDB types to Oracle NoSQL types for more details.
- Mandatory (Y/N): N
- **Example:** "DDBSortKey" : "Skey:STRING"

(i) Note

If the sort key contains dash(-) or dot(.), Migrator will replace it with underscore(_) as NoSQL column name does not support dot and dash.

overwrite

 Purpose: Indicates the behavior of NoSQL Database Migrator when the record being migrated from the source is already present in the sink.

If the value is set to false, when migrating tables the NoSQL Database Migrator skips those records for which the same primary key already exists in the sink.

If the value is set to true, when migrating tables the NoSQL Database Migrator overwrites those records for which the same primary key already exists in the sink.

If not specified, it defaults to true.

- Data Type: boolean
- Mandatory (Y/N): N



• Example: "overwrite" : false

Oracle NoSQL Database Cloud Service

The configuration file format for Oracle NoSQL Database Cloud Service as a sink of NoSQL Database Migrator is shown below.

Sink Configuration Template

```
"sink" : {
  "type" : "nosqldb_cloud",
  "endpoint" : "<Oracle NoSQL Cloud Service Endpoint>",
  "table" : "",
  "compartment" : "<OCI compartment name or id>",
  "includeTTL": <true | false>,
  "ttlRelativeDate" : "<date-to-use in UTC>",
  "schemaInfo" : {
   "schemaPath" : "</path/to/a/schema/file>",
   "defaultSchema" : <true | false>,
   "useSourceSchema" : <true | false>,
   "DDBPartitionKey" : <"name:type">,
   "DDBSortKey" : "<name:type>",
   "onDemandThroughput" : <true | false>,
   "readUnits" : ,
   "writeUnits" : ,
   "storageSize" : <storage size in GB>
  },
  "credentials" : "</path/to/oci/credential/file>",
  "credentialsProfile" : "<profile name in oci config file>",
  "useInstancePrincipal" : <true|false>,
  "useDelegationToken" : <true | false>,
  "useSessionToken" : <true | false>,
  "useOKEWorkloadIdentity" : <true | false>,
  "writeUnitsPercent" : ,
  "requestTimeoutMs" : <timeout in milli seconds>,
  "overwrite" : <true | false>
```

Sink Parameters

Common Configuration Parameters

```
type
Use "type" : "nosqldb_cloud"

endpoint
Example:
    Region ID: "endpoint" : "us-ashburn-1"
    URL format: "endpoint" : "https://objectstorage.us-ashburn-1.oraclecloud.com"

credentials
Example:
1. "credentials" : "/home/user/.oci/config"
```

"credentials" : "/home/user/security/config"



credentialsProfile

Example:

- 1. "credentialsProfile" : "DEFAULT"
- 2. "credentialsProfile" : "ADMIN_USER"
- useInstancePrincipal

Example: "useInstancePrincipal" : true

• <u>useDelegationToken</u>

Example: "useDelegationToken" : true



The authentication with delegation token is supported only when the NoSQL Database Migrator is running from a Cloud Shell.

useOKEWorkloadIdentity

Example: "useOKEWorkloadIdentity" : true

useSessionToken

Example: "useSessionToken" : true

<u>requestTimeoutMs</u>

Example: "requestTimeoutMs" : 5000

Unique Configuration Parameter

- table
- compartment
- includeTTL
- ttlRelativeDate
- schemalnfo
- schemaInfo.schemaPath
- schemaInfo.defaultSchema
- schemaInfo.useSourceSchema
- schemaInfo.DDBPartitionKey
- schemaInfo.DDBSortKey
- schemalnfo.onDemandThroughput
- schemaInfo.readUnits
- schemaInfo.writeUnits
- schemaInfo.storageSize
- writeUnitsPercent
- <u>overwrite</u>

table

Purpose: Specifies the table name to store the migrated data.



You must ensure that this table exists in your Oracle NoSQL Database Cloud Service. Otherwise, you have to use the schemaInfo object in the sink configuration to instruct the NoSQL Database Migrator to create the table.

The schema of this table must match the source data.

- Data Type: string
- Mandatory (Y/N): Y
- **Example:**
 - To specify a table "table" : "mytable"
 - To specify a child table "table" : "mytable.child"

(i) Note

You can migrate the child tables from a valid data source to Oracle NoSQL Database Cloud Service. The NoSQL Database Migrator copies only a single table in each execution. Ensure that the parent table is migrated before the child table.

compartment

Purpose: Specifies the name or OCID of the compartment in which the table resides.

If you do not provide any value, it defaults to the *root* compartment.

You can find your compartment's OCID from the Compartment Explorer window under Governance in the OCI Cloud Console.

- Data Type: string
- Mandatory (Y/N): Y, if the table is not in the root compartment of the tenancy OR when the useInstancePrincipal parameter is set to true.



(i) Note

If the useInstancePrincipal parameter is set to true, the compartment must specify the compartment OCID and not the name.

Example:

Compartment name

```
"compartment" : "mycompartment"
```

Compartment name qualified with its parent compartment

```
"compartment" : "parent.childcompartment"
```

No value provided. Defaults to the root compartment.

```
"compartment": ""
```

Compartment OCID

```
"compartment" : "ocid1.tenancy.oc1...4ksd"
```



includeTTL

 Purpose: Specifies whether or not to include TTL metadata for table rows provided by the source when importing Oracle NoSQL Database tables.

If you do not specify this parameter, it defaults to false. In that case, the NoSQL Database Migrator does not include TTL metadata for table rows provided by the source when importing Oracle NoSQL Database tables.

If set to true, the NoSQL Database Migrator tool performs the following checks on the TTL metadata when importing a table row:

- If you import a row that does not have _metadata definition, the NoSQL Database
 Migrator tool sets the TTL to 0, which means the row never expires.
- If you import a row that has _metadata definition, the NoSQL Database Migrator tool compares the TTL value against a Reference Time when a row gets imported. If the row has already expired relative to the Reference Time, then it is skipped. If the row has not expired, then it is imported along with the TTL value. By default, the Reference Time of import operation is the current time in milliseconds, obtained from System.currentTimeMillis(), of the machine where the NoSQL Database Migrator tool is running. But you can also set a custom Reference Time using the ttlRelativeDate configuration parameter if you want to extend the expiration time and import rows that would otherwise expire immediately.

The formula to calculate the expiration time of a row is as follows:

```
expiration = (TTL value of source row in milliseconds - Reference Time
in milliseconds)
if (expiration <= 0) then it indicates that row has expired.</pre>
```

(i) Note

Since Oracle NoSQL TTL boundaries are in hours and days, in some cases, the TTL of the imported row might get adjusted to the nearest hour or day. For example, consider a row that has expiration value of 1629709200000 ($2021-08-23\ 09:00:00$) and Reference Time value is 1629707962582 ($2021-08-23\ 08:39:22$). Here, even though the row is not expired relative to the Reference Time when this data gets imported, the new TTL for the row is $1629712800000\ (2021-08-23\ 10:00:00)$.

- Data Type: boolean
- Mandatory (Y/N): N
- **Example:** "includeTTL" : true

ttlRelativeDate

Purpose: Specify a UTC date in the YYYY-MM-DD hh:mm:ss format used to set the TTL expiry of table rows during importing into the Oracle NoSQL Database.

If a table row in the data you are exporting has expired, you can set the ttlRelativeDate parameter to a date before the expiration time of the table row in the exported data.

If you do not specify this parameter, it defaults to the current time in milliseconds, obtained from System.currentTimeMillis(), of the machine where the NoSQL Database Migrator tool is running.



- Data Type: date
- Mandatory (Y/N): N
- **Example:** "ttlRelativeDate" : "2021-01-03 04:31:17"

Let us consider a scenario where table rows expire after seven days from 1-Jan-2021. After exporting this table, on 7-Jan-2021, you run into an issue with your table and decide to import the data. The table rows are going to expire in one day (data expiration date minus the default value of ttlRelativedate configuration parameter, which is the current date). But if you want to extend the expiration date of table rows to five days instead of one day, use the ttlRelativeDate parameter and choose an earlier date. Therefore, in this scenario if you want to extend expiration time of the table rows by five days, set the value of ttlRelativeDate configuration parameters to 3-Jan-2021, which is used as Reference Time when table rows get imported.

schemalnfo

Purpose: Specifies the schema for the data being migrated.

If you do not specify this parameter, the NoSQL Database Migrator assumes that the table already exists in your Oracle NoSQL Database Cloud Service.

If this parameter is not specified and the table does not exist in the sink, the migration fails.

- Data Type: Object
- Mandatory (Y/N): N

schemalnfo.schemaPath

Purpose: Specifies the absolute path to a file containing DDL statements for the NoSQL table.

The NoSQL Database Migrator executes the DDL commands listed in this file before migrating the data.

The NoSQL Database Migrator does not support more than one DDL statement per line in the schemaPath file.

- Data Type: string
- Mandatory (Y/N): N



defaultSchema and schemaPath are mutually exclusive.

• **Example:** "schemaPath" : "/home/user/schema_file"

schemalnfo.defaultSchema

- Purpose: Setting this parameter to Yes instructs the NoSQL Database Migrator to create a
 table with default schema. The default schema is defined by the migrator itself. For more
 information about default schema definitions, see *Default Schema* in Using Oracle NoSQL
 Data Migrator.
- Data Type: boolean
- Mandatory (Y/N): N





defaultSchema and schemaPath are mutually exclusive.

schemalnfo.useSourceSchema

- Purpose: Specifies whether or not the sink uses the table schema definition provided by the source when migrating NoSQL tables.
- Data Type: boolean
- Mandatory (Y/N): N

Note

defaultSchema, schemaPath, and useSourceSchema parameters are mutually exclusive. Specify only one of these parameters.

Example:

With Default Schema:

```
"schemaInfo": {
  "defaultSchema": true,
  "readUnits": 100,
  "writeUnits": 60,
  "storageSize": 1
}
```

With a pre-defined schema:

```
"schemaInfo": {
   "schemaPath": "<complete/path/to/the/schema/definition/file>",
   "readUnits": 100,
   "writeUnits": 100,
   "storageSize": 1
}
```

With source schema:

```
"schemaInfo": {
   "useSourceSchema": true,
   "readUnits": 100,
   "writeUnits": 60,
   "storageSize": 1
}
```

schemaInfo.DDBPartitionKey

• Purpose: Specifies the DynamoDB partition key and the corresponding Oracle NoSQL Database type to be used in the sink Oracle NoSQL Database table. This key will be used as a NoSQL DB table shard key. This is applicable only when defaultSchema is set to true and the source format is dynamodb_json. See Mapping of DynamoDB types to Oracle NoSQL types for more details.



- Mandatory (Y/N): Y, if defaultSchema is true and the source is dynamodb_json.
- Example: "DDBPartitionKey" : "PersonID:INTEGER"

Note

If the partition key contains dash(-) or dot(.), Migrator will replace it with underscore(_) as NoSQL column name does not support dot and dash.

schemalnfo.DDBSortKey

- **Purpose:** Specifies the DynamoDB sort key and its corresponding Oracle NoSQL Database type to be used in the target Oracle NoSQL Database table. If the importing DynamoDB table does not have a sort key, this attribute must not be set. This key will be used as a non-shard portion of the primary key in the NoSQL DB table. This is applicable only when defaultSchema is set to true and the source is dynamodb_json. See Mapping of DynamoDB types to Oracle NoSQL types for more details.
- Mandatory (Y/N): N
- **Example:** "DDBSortKey" : "Skey:STRING"

Note

If the sort key contains dash(-) or dot(.), Migrator will replace it with underscore(_) as NoSQL column name does not support dot and dash.

schemainfo.onDemandThroughput

Purpose: Specifies to create the table with on-demand read and write throughput. If this
parameter is not set, the table is created with provisioned capacity.
 The default value is false.

Note

This parameter is not applicable for child tables as they share the throughput of the top-level parent table.

- Data Type: Boolean
- Mandatory (Y/N): N
- Example: "onDemandThroughput" : "true"

schemaInfo.readUnits

• **Purpose:** Specifies the read throughput of the new table.



(i) Note

- This parameter is not applicable for tables provisioned with on-demand capacity.
- This parameter is not applicable for child tables as they share the read throughput of the top-level parent table.
- Data Type: integer
- Mandatory (Y/N): Y, if the table is not a child table or if schemaInfo.onDemandThroughput parameter is set to false, else N.
- Example: "readUnits" : 100

schemaInfo.writeUnits

• **Purpose:** Specifies the write throughput of the new table.

(i) Note

- This parameter is not applicable for tables provisioned with on-demand capacity.
- This parameter is not applicable for child tables as they share the write throughput of the top-level parent table.
- Data Type: integer
- Mandatory (Y/N): Y, if the table is not a child table or if schemaInfo.onDemandThroughput parameter is set to false, else N.
- **Example:** "writeUnits" : 100

schemaInfo.storageSize

Purpose: Specifies the storage size of the new table in GB.

Note

This parameter is not applicable for child tables as they share the storage size of the top-level parent table.

- Data Type: integer
- Mandatory (Y/N): Y, if the table is not a child table, else N.
- Example:
 - With schemaPath

```
"schemaInfo" : {
  "schemaPath" : "</path/to/a/schema/file>",
  "readUnits" : 500,
  "writeUnits" : 1000,
  "storageSize" : 5 }
```



With defaultSchema

```
"schemaInfo" : {
  "defaultSchema" : Yes,
  "readUnits" : 500,
  "writeUnits" : 1000,
  "storageSize" : 5
}
```

writeUnitsPercent

• **Purpose:** Specifies the Percentage of table write units to be used during the migration activity. The amount of time required to migrate data is directly proportional to this attribute.

The default value is 90. The valid range is any integer between 1 to 100.

See <u>Troubleshooting the Oracle NoSQL Database Migrator</u> to learn how to use this attribute to improve the data migration speed.

Data Type: integer

Mandatory (Y/N): N

Example: "writeUnitsPercent" : 90

overwrite

 Purpose: Indicates the behavior of NoSQL Database Migrator when the record being migrated from the source is already present in the sink.

If the value is set to false, when migrating tables the NoSQL Database Migrator skips those records for which the same primary key already exists in the sink.

If the value is set to true, when migrating tables the NoSQL Database Migrator overwrites those records for which the same primary key already exists in the sink.

If not specified, it defaults to true.

Data Type: boolean

Mandatory (Y/N): N

• Example: "overwrite" : false

Transformation Configuration Templates

This topic explains the configuration parameters for the different transformations supported by the Oracle NoSQL Database Migrator. For the complete configuration file template, see **Configuration File** in Terminology used with Oracle NoSQL Database Migrator.

Oracle NoSQL Database Migrator lets you modify the data, that is, add data transformations as part of the migration activity. You can define multiple transformations in a single migration. In such a case, the order of transformations is vital because the source data undergoes each transformation in the given order. The output of one transformation becomes the input to the next one in the migrator pipeline.

The different transformations supported by the NoSQL Data Migrator are:



Table 1-3 Transformations

Transformation Config Attribute	You can use this transformation to	
ignoreFields	Ignore the identified columns from the source row before writing to the sink.	
includeFields	Include the identified columns from the source row before writing to the sink.	
renameFields	Rename the identified columns from the source row before writing to the sink.	
aggregateFields	Aggregate multiple columns from the source into a single column in the sink. As part of this transformation, you can also identify the columns that you want to exclude in the aggregation. Those fields will be skipped from the aggregated column.	

You can find the configuration template for each supported transformation below.

ignoreFields

The configuration file format for the ignoreFields transformation is shown below.

Transformation Configuration Template

```
"transforms" : {
  "ignoreFields" : ["<field1>","<field2>",...]
```

Transformation Parameter

ignoreFields

Purpose: An array of the column names to be ignored from the source records.



(i) Note

You can supply only top-level fields. Transformations can not be applied on the data in the nested fields.

- Data Type: array of strings
- Mandatory (Y/N): Y
- **Example:** To ignore the columns named "name" and "address" from the source record:

```
"ignoreFields" : ["name", "address"]
```



includeFields

The configuration file format for the includeFields transformation is shown below.

Transformation Configuration Template

```
"transforms" : {
  "includeFields" : ["<field1>","<field2>",...]
```

Transformation Parameter

includeFields

Purpose: An array of the column names to be included from the source records. It only includes the fields specified in the array, the rest of the fields are ignored.



(i) Note

The NoSQL Database Migrator tool throws an error if you specify an empty array. Additionally, you can specify only the top-level fields. The NoSQL Database Migrator tool does not apply transformations to the data in the nested fields.

- Data Type: array of strings
- Mandatory (Y/N): Y
- **Example:** To include the columns named "age" and "gender" from the source record:

```
"includeFields" : ["age", "gender"]
```

renameFields

The configuration file format for the renameFields transformation is shown below.

Transformation Configuration Template

```
"transforms" : {
  "renameFields" : {
    "<old_name>" : "<new_name>",
    "<old_name>" : "<new_name>,"
    . . . . .
```

Transformation Parameter

renameFields

Purpose: Key-Value pairs of the old and new names of the columns to be renamed.





(i) Note

You can supply only top-level fields. Transformations can not be applied on the data in the nested fields.

- Data Type: JSON object
- Mandatory (Y/N): Y
- Example: To rename the column named "residence" to "address" and the column named " id" to "id":

```
"renameFields" : { "residence" : "address", "_id" : "id" }
```

aggregateFields

The configuration file format for the aggregateFields transformation is shown below.

Transformation Configuration Template

```
"transforms" : {
  "aggregateFields" : {
    "fieldName" : "name of the new aggregate field",
    "skipFields" : ["<field1>","<field2">,...]
}
```

Transformation Parameter

aggregateFields

- **Purpose:** Name of the aggregated field in the sink.
- Data Type: string
- Mandatory (Y/N): Y
- **Example:** If the given record is:

```
"id" : 100,
"name" : "john",
"address" : "USA",
"age" : 20
```

If the aggregate transformation is:

```
"aggregateFields" : {
 "fieldName" : "document",
 "skipFields" : ["id"]
```



The aggregated column in the sink looks like:

```
"id": 100,
"document": {
  "name": "john",
  "address": "USA",
  "age": 20
```

Mapping of DynamoDB table to Oracle NoSQL table

In DynamoDB, a table is a collection of items, and each item is a collection of attributes. Each item in the table has a unique identifier, or a primary key. Other than the primary key, the table is schema-less. Each item can have its own distinct attributes.

DynamoDB supports two different kinds of primary keys:

- Partition key A simple primary key, composed of one attribute known as the partition key. DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function determines the partition in which the item will be stored.
- Partition key and sort key As a composite primary key, this type of key is composed of two attributes. The first attribute is the partition key, and the second attribute is the sort key. DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition in which the item will be stored. All items with the same partition key value are stored together, in sorted order by sort key value.

In contrast, Oracle NoSQL tables support flexible data models with both schema and schemaless design.

There are two different ways of modelling a DynamoDB table:

1. Modeling DynamoDB table as a JSON document(Recommended): In this modeling, you map all the attributes of the Dynamo DB tables into a JSON column of the NoSQL table except partition key and sort key. You will model partition key and sort key as the Primary Key columns of the NoSQL table. You will use AggregateFields transform in order to aggregate non-primary key data into a JSON column.



(i) Note

The Migrator provides a user-friendly configuration defaultSchema to automatically create a schema-less DDL table which also aggregates attributes into a JSON column.

Modeling DynamoDB table as fixed columns in NoSQL table: In this modeling, for each attribute of the DynamoDB table, you will create a column in the NoSQL table as specified in the Mapping of DynamoDB types to Oracle NoSQL types. You will model partition key and sort key attributes as Primary key(s). This should be used only when you are certain that importing DynamoDB table schema is fixed and each item has values for the most of the attributes. If DynamoDB items do not have common attributes, this can result in lot of NoSQL columns with empty values.





Note

We highly recommend using schema-less tables when migrating data from DynamoDB to Oracle NoSQL Database due to the nature of DynamoDB tables being schema-less. This is especially for large tables where the content of each record may not be uniform across the table.

Oracle NoSQL to Parquet Data Type Mapping

Describes the mapping of Oracle NoSQL data types to Parquet data types.

NoSQL Type	Parquet Type	
BOOLEAN	BOOLEAN	
INTEGER	INT32	
LONG	INT64	
FLOAT	DOUBLE	
DOUBLE	DOUBLE	
BINARY	BINARY	
FIXED_BINARY	BINARY	
STRING	BINARY(STRING)	
ENUM	BINARY(STRING)	
	or	
	BINARY(ENUM), if the logical ENUM is configured	
UUID	BINARY(STRING)	
	or	
	FIXED_BINARY(16), if the logical UUID is	
	configured	
TIMESTAMP(p)	INT64(TIMESTAMP(p))	
NUMBER	DOUBLE	
field_name ARRAY(T)		
	group field_name(LIST) {	
	<pre>repeated group list {</pre>	
	repeated group list { required T element	
	required T element	
	required T element	
	required T element	
field_name MAP(T)	required T element } }	
field_name MAP(T)	<pre>required T element } group field_name (MAP) {</pre>	
field_name MAP(T)	required T element } group field_name (MAP) { repeated group key_value	
field_name MAP(T)	required T element } group field_name (MAP) { repeated group key_value (MAP_KEY_VALUE) {	
field_name MAP(T)	<pre>required T element } group field_name (MAP) { repeated group key_value (MAP_KEY_VALUE) { required binary key (STRING);</pre>	
field_name MAP(T)	<pre>required T element } group field_name (MAP) { repeated group key_value (MAP_KEY_VALUE) { required binary key (STRING); required T value;</pre>	
field_name MAP(T)	<pre>required T element } group field_name (MAP) { repeated group key_value (MAP_KEY_VALUE) { required binary key (STRING); required T value; }</pre>	
field_name MAP(T)	<pre>required T element } group field_name (MAP) { repeated group key_value (MAP_KEY_VALUE) { required binary key (STRING); required T value;</pre>	



NoSQL Type	Parquet Type
field_name RECORD(K T N , K T N ,) where:	group field_name {
K = Key name	ni == true ? optional Ti ki :
T = Type	required Ti ki
N = Nullable or not	}
JSON	BINARY(STRING)
	or
	BINARY(JSON), if logical JSON is configured

(i) Note

When the NoSQL Number type is converted to Parquet Double type, there may be some loss of precision in case the value cannot be represented in Double. If the number is too big to represent as Double, it is converted to Double.NEGATIVE_INFINITY or Double.POSITIVE_INFINITY.

Mapping of DynamoDB types to Oracle NoSQL types

The table below shows the mapping of DynamoDB types to Oracle NoSQL types.

Table 1-4 Mapping DynamoDB type to Oracle NoSQL type

#	DynamoDB type	JSON type for NoSQL JSON column	Oracle NoSQL type
1	String (S)	JSON String	STRING
2	Number Type (N)	JSON Number	INTEGER/LONG/ FLOAT/DOUBLE/ NUMBER
3	Boolean (BOOL)	JSON Boolean	BOOLEAN
4	Binary type (B) - Byte buffer	BASE-64 encoded JSON String	BINARY
5	NULL	JSON null	NULL
6	String Set (SS)	JSON Array of Strings	ARRAY(STRING)
7	Number Set (NS)	JSON Array of Numbers	ARRAY(INTEGER/ LONG/FLOAT/DOUBLE/ NUMBER)
8	Binary Set (BS)	JSON Array of Base-64 encoded Strings	ARRAY(BINARY)
9	LIST (L)	Array of JSON	ARRAY(JSON)
10	MAP (M)	JSON Object	JSON
11	PARTITION KEY	NA	PRIMARY KEY and SHARD KEY
12	SORT KEY	NA	PRIMARY KEY
13	Attribute names with dash and dot	JSON field names with a underscore	Column names with underscore

Few additional points to consider while mapping DynamoDB types to Oracle NoSQL types:



- DynamoDB Supports only one data type for Numbers and can have up to 38 digits of
 precision, on contrast Oracle NoSQL supports many types to choose from based on the
 range and precision of the data. You can select the appropriate Number type that fits the
 range of your input data. If you are not sure of the nature of the data, NoSQL NUMBER
 type can be used.
- DynamoDB Supports only one data type for Numbers and can have up to 38 digits of
 precision, on contrast Oracle NoSQL supports many types to choose from based on the
 range and precision of the data. You can select the appropriate Number type that fits the
 range of your input data. If you are not sure of the nature of the data, NoSQL NUMBER
 type can be used.
- Partition key in DynamoDB has a limit of 2048 bytes but Oracle NoSQL Cloud Service has a limit of 64 bytes for the Primary key/Shard key.
- Sort key in DynamoDB has a limit of 1024 bytes but Oracle NoSQL Cloud Service has a limit of 64 bytes for the Primary key.
- Attribute names in DynamoDB can be 64KB long but Oracle NoSQL Cloud service column names have a limit of 64 characters.

Use Case Demonstrations

Learn how to perform data migration using the Oracle NoSQL Database Migrator for specific use cases. You can find detailed systematic instructions with code examples to perform migration in each of the use cases listed below.

Topics:

- Migrate from Oracle NoSQL Database to a JSON file
- Migrate from Oracle NoSQL Database On-Premise to Oracle NoSQL Database Cloud Service
- Migrate from JSON file source to Oracle NoSQL Database
- Migrate from MongoDB JSON file to Oracle NoSQL Database
- Migrate from DynamoDB JSON file to Oracle NoSQL Database
- Migrate from DynamoDB JSON file in AWS S3 to Oracle NoSQL Database
- Migrate from CSV file to Oracle NoSQL Database
- Migrate from Oracle NoSQL Database to OCI Object Storage Using Session Token Authentication

Migrate from Oracle NoSQL Database to a JSON file

This example shows how to use the Oracle NoSQL Database Migrator to copy data and the schema definition of a NoSQL table from Oracle NoSQL Database to a JSON file.

Use Case

An organization decides to train a model using the Oracle NoSQL Database data to predict future behaviors and provide personalized recommendations. They can take a periodic copy of the NoSQL Database tables data to a JSON file and apply it to the analytic engine to analyze and train the model. Doing this helps them separate the analytical queries from the low-latency critical paths.



Example

For the demonstration, let us look at how to migrate the data and schema definition of a NoSQL table called myTable from NoSQL Database to a JSON file.

Prerequisites

- Identify the source and sink for the migration.
 - Source: Oracle NoSQL Database
 - Sink: JSON file
- Identify the following details for the on-premise KVStore:
 - storeName: kvstore
 - helperHosts: <hostname>:5000
 - table: myTable

Procedure

To migrate the data and schema definition of myTable from Oracle NoSQL Database to a JSON file:

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 2. To generate the configuration file using the NoSQL Database Migrator, run the runMigrator command without any runtime parameter.

```
[~/nosqlMigrator]$./runMigrator
```

3. As you did not provide the configuration file as a runtime parameter, the utility prompts if you want to generate the configuration now. Type y.

```
Configuration file is not provided. Do you want to generate configuration? (y/n) [n]: y Generating a configuration file interactively.
```

4. Based on the prompts from the utility, choose your options for the Source configuration.

```
Enter a location for your config [./migrator-config.json]: /home/<user>/
nosqlMigrator/
Select the source:
1) nosqldb
2) nosqldb_cloud
3) file
4) object_storage_oci
5) aws_s3
#? 1

Configuration for source type=nosqldb

Enter store name of the Oracle NoSQL Database: kvstore

Enter comma separated list of host:port of Oracle NoSQL Database: <hostname>:5000
```



```
Enter fully qualified table name: myTable

Include TTL data? If you select 'yes' TTL of rows will also be included in the exported data.(y/n) [n]:

Is the store secured? (y/n) [y]:

Enter store operation timeout in milliseconds. (1-30000) [5000]:
```

5. Based on the prompts from the utility, choose your options for the Sink configuration.

```
Select the sink:
1) nosqldb
2) nosqldb_cloud
3) file
#? 3
Configuration for sink type=file
Select the sink file format:
1) json
2) parquet
#? 1
Enter path to a directory to store JSON data: /home/<user>/nosqlMigrator
would you like to export data to multiple files for each source?(y/n) [y]:
Would you like to store JSON in pretty format? (y/n) [n]: y
Would you like to migrate the table schema also? (y/n) [y]: y
Enter path to a file to store table schema: /home/<user>/nosqlMigrator/
myTableSchema.ddl
```

6. Based on the prompts from the utility, choose your options for the source data transformations. The default value is n.

```
Would you like to add transformations to source data? (y/n) [n]:
```

7. Enter your choice to determine whether to proceed with the migration in case any record fails to migrate.

```
Would you like to continue migration in case of any record/row is failed to migrate?: (y/n) [n]:
```

8. The utility displays the generated configuration on the screen.

```
Generated configuration is:
{
  "source" : {
    "type" : "nosqldb",
    "storeName" : "kvstore",
    "helperHosts" : ["<hostname>:5000"],
    "table" : "myTable",
```



```
"includeTTL" : true,
  "requestTimeoutMs" : 5000

},

"sink" : {
  "type" : "file",
  "format" : "json",
  "useMultiFiles" : false,
  "schemaPath" : "/home/<username>/nosqlMigrator/myTableSchema.ddl",
  "pretty" : true,
  "dataPath" : "/home/<username>/nosqlMigrator/"
},
  "abortOnError" : true,
  "migratorVersion" : "1.7.0"
}
```

Verification

To verify the migration, you can navigate to the specified sink directory and view the schema and data.

```
-- Exported myTable Data. JSON files are created in the supplied data path
[~/nosqlMigrator]$cat myTable_1_5.json
  "id" : 2,
  "document" : "{\"course\":\"Bio-
Technology\",\"name\":\"Raja\",\"studentid\":108}"
  "id" : 7,
  "document" : "{\"course\":\"Computer
Science\",\"name\":\"Ruby\",\"studentid\":100}"
  "id" : 4,
  "document" : "{\"course\":\"Computer
Science\",\"name\":\"Ruby\",\"studentid\":100}"
  "id" : 10,
  "document" : "{\"course\":\"Computer
Science\",\"name\":\"Neena\",\"studentid\":105}"
 "id" : 5,
  "document" :
"{\"course\":\"Journalism\",\"name\":\"Rani\",\"studentid\":106}"
[~/nosqlMigrator]$cat myTable_6_10.json
  "id" : 8,
  "document" : "{\"course\":\"Computer
Science\",\"name\":\"Tom\",\"studentid\":103}"
```



```
"id" : 3,
  "document" : "{\"course\":\"Computer
Science\",\"name\":\"John\",\"studentid\":107}"
  "id" : 9,
  "document" : "{\"course\":\"Computer
Science\",\"name\":\"Peter\",\"studentid\":109}"
  "id" : 6,
  "document" : "{\"course\":\"Bio-
Technology\",\"name\":\"Rekha\",\"studentid\":104}"
  "id" : 1,
  "document":
"{\"course\":\"Journalism\",\"name\":\"Tracy\",\"studentid\":110}"
-- Exported myTable Schema
[~/nosqlMigrator]$cat myTableSchema
CREATE TABLE IF NOT EXISTS myTable (id INTEGER, document JSON, PRIMARY
KEY(SHARD(id)))
```

Migrate from Oracle NoSQL Database On-Premise to Oracle NoSQL Database Cloud Service

This example shows how to use the Oracle NoSQL Database Migrator to copy data and the schema definition of a NoSQL table from Oracle NoSQL Database to Oracle NoSQL Database Cloud Service (NDCS).

Use Case

As a developer, you are exploring options to avoid the overhead of managing the resources, clusters, and garbage collection for your existing NoSQL Database KVStore workloads. As a solution, you decide to migrate your existing on-premise KVStore workloads to Oracle NoSQL Database Cloud Service because NDCS manages them automatically.

Example

For the demonstration, let us look at how to migrate the data and schema definition of a NoSQL table called myTable from the NoSQL Database KVStore to NDCS. We will also use this use case to show how to run the runMigrator utility by passing a precreated configuration file.

Prerequisites

- Identify the source and sink for the migration.
 - Source: Oracle NoSQL Database
 - Sink: Oracle NoSQL Database Cloud Service



 Identify your OCI cloud credentials and capture them in the OCI config file. Save the config file in /home/.oci/config. See Acquiring Credentials in Using Oracle NoSQL Database Cloud Service.

```
[DEFAULT]
tenancy=ocid1.tenancy.oc1....
user=ocid1.user.oc1....
fingerprint= 43:d1:....
key_file=</fully/qualified/path/to/the/private/key/>
pass_phrase=<passphrase>
```

- Identify the region endpoint and compartment name for your Oracle NoSQL Database Cloud Service.
 - endpoint: us-phoenix-1
 - compartment: developers
- Identify the following details for the on-premise KVStore:
 - storeName: kvstore
 - helperHosts: <hostname>:5000
 - table: myTable

Procedure

To migrate the data and schema definition of myTable from NoSQL Database KVStore to NDCS:

1. Prepare the configuration file (in JSON format) with the identified Source and Sink details. See Source Configuration Templates and Sink Configuration Templates.

```
"source" : {
    "type" : "nosqldb",
    "storeName" : "kvstore",
    "helperHosts" : ["<hostname>:5000"],
    "table" : "myTable",
    "requestTimeoutMs" : 5000
  "sink" : {
    "type" : "nosqldb cloud",
    "endpoint" : "us-phoenix-1",
    "table" : "myTable",
    "compartment" : "developers",
    "schemaInfo" : {
      "schemaPath" : "<complete/path/to/the/JSON/file/with/DDL/
commands/for/the/schema/definition>",
      "readUnits" : 100,
      "writeUnits" : 100,
      "storageSize" : 1
    },
    "credentials" : "<complete/path/to/oci/config/file>",
    "credentialsProfile" : "DEFAULT",
    "writeUnitsPercent" : 90,
    "requestTimeoutMs" : 5000
  },
  "abortOnError" : true,
```



```
"migratorVersion" : "1.7.0"
}
```

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing the configuration file using the --config or -c option.

```
[~/nosqlMigrator/nosql-migrator-1.7.0]$./runMigrator --config <complete/path/to/the/JSON/config/file>
```

4. The utility proceeds with the data migration, as shown below.

```
Records provided by source=10, Records written to sink=10, Records failed=0.

Elapsed time: Omin 10sec 426ms

Migration completed.
```

Validation

To validate the migration, you can login to your NDCS console and verify that myTable is created with the source data.

Migrate from JSON file source to Oracle NoSQL Database

This example shows the usage of Oracle NoSQL Database Migrator to copy data from a JSON file source to Oracle NoSQL Database.

Use Case

After evaluating multiple options, an organization finalizes Oracle NoSQL Database as its NoSQL Database platform. As its source contents are in JSON file format, they are looking for a way to migrate them to Oracle NoSQL Database.

Example

In this example, you will learn to migrate the data from a JSON file called <code>SampleData.json</code>. You run the <code>runMigrator</code> utility by passing a pre-created configuration file. If the configuration file is not provided as a run time parameter, the <code>runMigrator</code> utility prompts you to generate the configuration through an interactive procedure.

Prerequisites

- Identify the source and sink for the migration.
 - Source: JSON source file.
 SampleData.json is the source file. It contains multiple JSON documents with one document per line, delimited by a new line character.

```
{"id":6,"val_json":{"array":
["q","r","s"],"date":"2023-02-04T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]],"nested":{"arrayofobjects":
[{"datefield":"2023-03-04T02:38:57.520Z","numfield":30,"strfield":"foo54"],
{"datefield":"2023-02-04T02:38:57.520Z","numfield":56,"strfield":"bar23"
}],"nestNum":10,"nestString":"bar"},"num":1,"string":"foo"}}
{"id":3,"val_json":{"array":
```



```
["q", "h", "i"], "date": "2023-02-02T02:38:57.520Z", "nestarray": [[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-02-02T02:38:57.520Z","numfield":28,"strfield":"foo3"
},
{"datefield":"2023-02-02T02:38:57.520Z","numfield":38,"strfield":"bar"}]
, "nestNum":10, "nestString": "bar" }, "num":1, "string": "foo" } }
{"id":7, "val json": { "array":
["a","b","c"],"date":"2023-02-20T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-01-20T02:38:57.520Z","numfield":28,"strfield":"foo"}
{"datefield":"2023-01-22T02:38:57.520Z","numfield":38,"strfield":"bar"}]
, "nestNum":10, "nestString":"bar"}, "num":1, "string":"foo"}}
{"id":4, "val_json": { "array":
["j","k","l"],"date":"2023-02-03T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-02-03T02:38:57.520Z","numfield":28,"strfield":"foo"}
{"datefield":"2023-02-03T02:38:57.520Z","numfield":38,"strfield":"bar"}]
, "nestNum":10, "nestString":"bar"}, "num":1, "string":"foo"}}
```

- Sink: Oracle NoSQL Database
- Identify the following details for the JSON source file:
 - schemaPath: <absolute path to the schema definition file containing DDL statements for the NoSQL table at the sink>.
 In this example, the DDL file is schema_json.ddl.

```
create table Migrate_JSON (id INTEGER, val_json JSON, PRIMARY KEY(id));
```

The Oracle NoSQL Database Migrator provides an option to create a table with the default schema if the schemaPath is not provided. For more details, see Identify the Source and Sink topic in the Workflow for Oracle NoSQL Database Migrator.

- Datapath: <absolute path to a file or directory containing the JSON data for migration>
- Identify the following details for the on-premise KVStore:
 - storename: kvstore
 - helperHosts: <hostname>:5000
 - table: Migrate_JSON

Procedure

To migrate the JSON source file from SampleData.json to Oracle NoSQL Database Cloud Service, perform the following:

1. Prepare the configuration file (in JSON format) with the identified source and sink details. See Source Configuration Templates and Sink Configuration Templates.

```
Generated configuration is:
{
    "source" : {
      "type" : "file",
      "format" : "json",
```



```
"schemaInfo" : {
    "schemaPath" : "/home/<username>/nosqlMigrator/schema json.ddl"
  "dataPath" : "/home/<username>/nosqlMigrator/SampleData.json"
"sink" : {
 "type" : "nosqldb",
  "storeName" : "kvstore",
  "helperHosts" : ["localhost:5000"],
  "table" : "Migrate_JSON",
  "includeTTL" : false,
  "schemaInfo" : {
    "useSourceSchema" : true
 },
  "overwrite" : true,
  "requestTimeoutMs" : 5000
},
"abortOnError" : true,
"migratorVersion" : "1.7.0"
```

- Open the command prompt and navigate to the directory where you extracted the Oracle NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing the configuration file using the --config or -c option.

```
[~/nosql-migrator-1.7.0]$./runMigrator --config <complete/path/to/the/config/file>
```

4. The utility proceeds with the data migration, as shown below. The Migrate_JSON table is created at the sink with the schema provided in the schemaPath.

```
creating source from given configuration:
[INFO] source creation completed
[INFO] creating sink from given configuration:
[INFO] sink creation completed
[NFO] creating migrator pipeline
[INFO] [nosqldb sink] : start loading DDLs
[nosqldb sink] : executing DDL: create table Migrate_JSON (id INTEGER,
val json JSON, PRIMARY KEY(id))
[INFO] [nosqldb sink] : completed loading DDLs
[INFO] migration started
[INFO] Start writing data to OnDB Sink
[INFO] executing for source:SampleData
[INFO] [json file source] : start parsing JSON records from file:
SampleData.json
[INFO] Writing data to OnDB Sink completed.
[INFO] migration completed.
Records provided by source=4, Records written to sink=4, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 81ms
Migration completed.
```



Verification

Start the SQL prompt in your data store.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the new table is created with the source data:

```
SELECT * FROM Migrate_JSON
```

Output:

```
{"id":7,"val_json":{"array":
["a", "b", "c"], "date": "2023-02-20T02:38:57.520Z", "nestarray": [[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-01-20T02:38:57.520Z","numfield":28,"strfield":"foo"},
{"datefield":"2023-01-22T02:38:57.520Z","numfield":38,"strfield":"bar"}],"nest
Num":10, "nestString": "bar" }, "num":1, "string": "foo" }}
{"id":3, "val_json": { "array":
["g","h","i"],"date":"2023-02-02T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-02-02T02:38:57.520Z","numfield":28,"strfield":"foo3"},
{"datefield":"2023-02-02T02:38:57.520Z","numfield":38,"strfield":"bar"}],"nest
Num":10, "nestString": "bar" }, "num":1, "string": "foo" }}
{"id":4, "val_json": { "array":
["j","k","l"],"date":"2023-02-03T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-02-03T02:38:57.520Z","numfield":28,"strfield":"foo"},
{"datefield":"2023-02-03T02:38:57.520Z","numfield":38,"strfield":"bar"}],"nest
Num":10, "nestString": "bar" }, "num":1, "string": "foo" }}
{"id":6, "val_json": { "array":
["q","r","s"],"date":"2023-02-04T02:38:57.520Z","nestarray":[[1,2,3],
[10,20,30]], "nested": { "arrayofobjects":
[{"datefield":"2023-03-04T02:38:57.520Z","numfield":30,"strfield":"foo54"},
{"datefield":"2023-02-04T02:38:57.520Z","numfield":56,"strfield":"bar23"}],"ne
stNum":10, "nestString": "bar" }, "num":1, "string": "foo" }}
```

4 rows returned

Migrate from MongoDB JSON file to Oracle NoSQL Database

This example shows how to use the Oracle NoSQL Database Migrator to copy MongoDB-formatted data to Oracle NoSQL Database.

Use Case

After evaluating multiple options, an organization finalizes Oracle NoSQL Database as its NoSQL Database platform. The tables and data are in MongoDB and the organization wants to migrate both to Oracle NoSQL Database.

You can copy a file or directory containing the MongoDB exported JSON data for migration by specifying the file or directory in the source configuration template.



A sample MongoDB-formatted JSON file is as follows:

```
{"_id":0,"name":"Aimee Zank","scores":
[{"score":1.463179736705023,"type":"exam"},
{"score":11.78273309957772,"type":"quiz"},
{"score":35.8740349954354,"type":"homework"}]}
{"_id":1, "name": "Aurelia Menendez", "scores":
[{"score":60.06045071030959,"type":"exam"},
{ "score":52.79790691903873, "type": "quiz" },
{"score":71.76133439165544,"type":"homework"}]}
{" id":2,"name":"Corliss Zuk","scores":
[{"score":67.03077096065002,"type":"exam"},
{"score":6.301851677835235,"type":"quiz"},
{"score":66.28344683278382,"type":"homework"}]}
{" id":3,"name":"Bao Ziglar","scores":
[{"score":71.64343899778332,"type":"exam"},
{"score":24.80221293650313,"type":"quiz"},
{"score":42.26147058804812,"type":"homework"}]}
{"_id":4, "name": "Zachary Langlais", "scores":
[{"score":78.68385091304332,"type":"exam"},
{"score":90.2963101368042,"type":"quiz"},
{"score":34.41620148042529,"type":"homework"}]}
```

MongoDB supports two types of extensions to the formatted JSON files, *Canonical mode* and *Relaxed mode*. You can supply the MongoDB-formatted JSON file that is generated using the *mongoexport* tool in either Canonical or Relaxed mode. NoSQL Database Migrator supports both the modes.

For more information on the MongoDB Extended JSON (v2) file, See mongoexport formats.

For more information on the generation of MongoDB-formatted JSON file, See mongoexport.

Example

For the demonstration, let us look at how to migrate a MongoDB-formatted JSON file to Oracle NoSQL Database. We will use a manually created configuration file for this example.

Prerequisites

- Identify the source and sink for the migration.
 - Source: MongoDB-formatted JSON File
 - Sink: Oracle NoSQL Database
- Extract the data from MongoDB using the mongoexport utility. See mongoexport for more information.

Procedure

To migrate the MongoDB-formatted JSON data to Oracle NoSQL Database, you can choose from one of the following options:

- Using a default schema
- Using a custom schema



Using a default schema

Prepare the configuration file (in JSON format) with the identified Source and Sink details.
 See Source Configuration Templates and Sink Configuration Templates.

Here, you set the defaultSchema configuration parameter to true. Therefore, NoSQL Database Migrator creates a table with the default schema at the sink.

```
"source" : {
  "type" : "file",
  "format" : "mongodb_json",
  "dataPath" : "<complete/path/to/the/MongoDB/Formatted/JSON/file>"
},
"sink" : {
  "type" : "nosqldb",
  "storeName" : "kvstore",
  "helperHosts" : ["phoenix126166:5000"],
  "table" : "mongoImport",
  "includeTTL" : true,
  "schemaInfo" : {
    "defaultSchema" : true
  },
  "overwrite" : true,
  "requestTimeoutMs" : 5000
"abortOnError" : true,
"migratorVersion" : "1.7.0"
```

The default schema for MongoDB-formatted JSON file source is as follows:

```
CREATE TABLE IF NOT EXISTS <tablename>(id STRING, document JSON,PRIMARY KEY(SHARD(id));
```

Where:

- tablename = value provided for the table attribute in the configuration.
- id = The _id value from each document of the MongoDB exported JSON source file.
- document = For each document in the MongoDB exported file, the contents excluding the _id field are aggregated into the document column.

(i) Note

If the table <tablename> already exists in Oracle NoSQL Database and you want to migrate data to the table using the defaultSchema configuration, you must ensure that the existing table has the ID column in lower case (id) and is of the type STRING.

Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.



3. Run the runMigrator command by passing the configuration file. Use --config or -c option.

```
$./runMigrator --config <complete/path/to/the/JSON/config/file>
```

4. The utility proceeds with the data migration, as shown below.

```
creating source from given configuration:
source creation completed
creating sink from given configuration:
sink creation completed
creating migrator pipeline
[nosqldb sink] : start loading DDLs
[nosqldb sink] : executing DDL: CREATE TABLE IF NOT EXISTS mongoImport (id
STRING, document JSON, PRIMARY KEY(SHARD(id)))
[nosqldb sink] : completed loading DDLs
migration started
Start writing data to OnDB Sink
executing for source:mongoDBSample
[mongo file source] : start parsing MongoDB JSON records from file:
mongoDBSample.json
Writing data to OnDB Sink completed.
migration completed.
Records provided by source=5, Records written to sink=5, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 108ms
Migration completed.
```

Verification

Start the SQL prompt in your KVStore.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the mongoImport table is created with the source data:

```
sql-> select * from mongoImport;
{ "id": "4", "document ": { "name": "Zachary Langlais", "scores":
[{"score":78.68385091304332,"type":"exam"},
{"score":90.2963101368042,"type":"quiz"},
{"score":34.41620148042529,"type":"homework"}]}}
{"id":"1", "document":{ "name": "Aurelia Menendez", "scores":
[{"score":60.06045071030959,"type":"exam"},
{"score":52.79790691903873,"type":"quiz"},
{"score":71.76133439165544,"type":"homework"}]}}
{ "id": "2", "document ": { "name": "Corliss Zuk", "scores":
[{"score":67.03077096065002,"type":"exam"},
{ "score":6.301851677835235, "type": "quiz" },
{ "score":66.28344683278382, "type": "homework" } ] } }
{ "id": "3", "document ": { "name": "Bao Ziglar", "scores":
[{"score":71.64343899778332,"type":"exam"},
{"score":24.80221293650313,"type":"quiz"},
{ "score":42.26147058804812, "type": "homework" } ] } }
{ "id": "0", "document ": { "name": "Aimee Zank", "scores":
```



```
[{"score":1.463179736705023,"type":"exam"},
{"score":11.78273309957772,"type":"quiz"},
{"score":35.8740349954354,"type":"homework"}]}}
```

Using a custom schema

Prepare the configuration file (in JSON format) with the identified Source and Sink details.
 See Source Configuration Templates and Sink Configuration Templates.

Here, you specify the file containing the DDL statement of the sink table in the schemaPath parameter of the source configuration template. Correspondingly, set the useSourceSchema configuration parameter to true in the sink configuration template.

You can generate a custom schema as follows:

- Note the names and data types for each column from the MongoDB-formatted JSON data. Use this information to create a schema DDL file for the Oracle NoSQL Database table.
- In the schema file, name the first column (primary key) as id of type STRING. Include
 the same name and type for the remaining columns as recorded in the MongoDBformatted JSON file.
- Save the schema file and note down it's complete path.

The following user-defined schema is used in this example:

```
CREATE TABLE IF NOT EXISTS sampleMongoDBImp (id STRING, name STRING, scores JSON, PRIMARY KEY(SHARD(id)));
```

You must include a renameFields transformation instructing NoSQL Database Migrator to convert the _id column to id while creating the table. For parameter details, see Transformation Configuration Templates. NoSQL Database Migrator creates a table with the custom schema at the sink.

```
"source" : {
  "type" : "file",
  "format" : "mongodb_json",
  "schemaInfo" : {
    "schemaPath" : "<complete/path/to/the/schema/file>"
  "dataPath" : "<complete/path/to/the/MongoDB/Formatted/JSON/file>"
},
"sink" : {
  "type" : "nosqldb",
  "storeName" : "kvstore",
  "helperHosts" : ["phoenix126166:5000"],
  "table" : "sampleMongoDBImp",
  "includeTTL" : true,
  "schemaInfo" : {
    "useSourceSchema" : true
  "overwrite" : false,
  "requestTimeoutMs" : 5000
```



```
},
"transforms": {
    "renameFields" : {
        "_id":"id"
    }
},
"abortOnError" : true,
"migratorVersion" : "1.7.0"
}
```

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing the configuration file. Use --config or -c option.
 - \$./runMigrator --config <complete/path/to/the/JSON/config/file>
- 4. The utility proceeds with the data migration, as shown below.

```
creating source from given configuration:
source creation completed
creating sink from given configuration:
sink creation completed
creating migrator pipeline
[nosqldb sink] : start loading DDLs
[nosqldb sink] : executing DDL: CREATE TABLE IF NOT EXISTS
sampleMongoDBImp (id INTEGER, name STRING, scores JSON, PRIMARY
KEY(SHARD(id)))
[nosqldb sink] : completed loading DDLs
migration started
Start writing data to OnDB Sink
executing for source:mongoDBSample
[mongo file source] : start parsing MongoDB JSON records from file:
mongoDBSample.json
Writing data to OnDB Sink completed.
migration completed.
Records provided by source=5, Records written to sink=5, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 72ms
Migration completed.
```

Verification

Start the SQL prompt in your KVStore.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the sampleMongoDBImp table is created with the source data:

```
sql-> select * from sampleMongoDBImp;
{"id":0,"name":"Aimee Zank","scores":
[{"score":1.463179736705023,"type":"exam"},
{"score":11.78273309957772,"type":"quiz"},
{"score":35.8740349954354,"type":"homework"}]}
{"id":2,"name":"Corliss Zuk","scores":
```



```
[{"score":67.03077096065002,"type":"exam"},
{"score":6.301851677835235,"type":"quiz"},
{ "score":66.28344683278382, "type": "homework" } ] }
{"id":1, "name": "Aurelia Menendez", "scores":
[{"score":60.06045071030959,"type":"exam"},
{"score":52.79790691903873,"type":"quiz"},
{ "score":71.76133439165544, "type": "homework" } ] }
{"id":3, "name": "Bao Ziglar", "scores":
[{"score":71.64343899778332,"type":"exam"},
{"score":24.80221293650313,"type":"quiz"},
{"score":42.26147058804812,"type":"homework"}]}
{"id":4, "name": "Zachary Langlais", "scores":
[{"score":78.68385091304332,"type":"exam"},
{"score":90.2963101368042,"type":"quiz"},
{"score":34.41620148042529,"type":"homework"}]}
5 rows returned
```

Migrate from DynamoDB JSON file to Oracle NoSQL Database

This example shows how to use Oracle NoSQL Database Migrator to copy DynamoDB JSON file to NoSQL Database.

Use Case:

After evaluating multiple options, an organization finalizes Oracle NoSQL Database over DynamoDB database. The organization wants to migrate their tables and data from DynamoDB to Oracle NoSQL Database (on-premises).

See Mapping of DynamoDB table to Oracle NoSQL table for more details.

You can migrate a file or directory containing the DynamoDB exported JSON data from a file system by specifying the path in the source configuration template.

A sample DynamoDB-formatted JSON File is as follows:

```
{"Item":{"Id":{"N":"101"},"Phones":{"L":[{"L":[{"S":"555-222"},
{"S":"123-567"}]}]},"PremierCustomer":{"BOOL":false},"Address":{"M":{"Zip":
{"N":"570004"},"Street":{"S":"21 main"},"DoorNum":{"N":"201"},"City":
{"S":"London"}}},"FirstName":{"S":"Fred"},"FavNumbers":{"NS":
["10"]},"LastName":{"S":"Smith"},"FavColors":{"SS":["Red","Green"]},"Age":
{"N":"22"},"ttl": {"N": "1734616800"}}}
{"Item":{"Id":{"N":"102"},"Phones":{"L":[{"L":
[{"S":"222-222"}]}]},"PremierCustomer":{"BOOL":false},"Address":{"M":{"Zip":
{"N":"560014"},"Street":{"S":"32 main"},"DoorNum":{"N":"1024"},"City":
{"S":"Wales"}}},"FirstName":{"S":"John"},"FavNumbers":{"NS":
["10"]},"LastName":{"S":"White"},"FavColors":{"SS":["Blue"]},"Age":
{"N":"48"},"ttl": {"N": "1734616800"}}}
```

You copy the exported DynamoDB table data from AWS S3 storage to a local mounted file system.



Example:

For this demonstration, you will learn how to migrate a DynamoDB JSON file to Oracle NoSQL Database (on-premises). You will use a manually created configuration file for this example.

Prerequisites

- Identify the source and sink for the migration.
 - Source: DynamoDB JSON File
 - Sink: Oracle NoSQL Database (on-premises)
- In order to import DynamoDB table data to Oracle NoSQL Database, you must first export the DynamoDB table to S3. See the steps provided in Exporting DynamoDB table data to Amazon S3 to export your table. While exporting, you select the format as DynamoDB JSON. The exported data contains DynamoDB table data in multiple gzip files as shown below.

```
/ 01639372501551-bb4dd8c3
|-- 01639372501551-bb4dd8c3 ==> exported data prefix
|----data
|----sxz3hjr3re2dzn2ymgd2gi4iku.json.gz ==>table data
|----manifest-files.json
|----manifest-files.md5
|----manifest-summary.json
|----manifest-summary.md5
|----_started
```

 You must download the files from AWS S3. The structure of the files after the download will be as shown below.

```
download-dir/01639372501551-bb4dd8c3
|----data
|----sxz3hjr3re2dzn2ymgd2gi4iku.json.gz ==>table data
|----manifest-files.json
|---manifest-files.md5
|---manifest-summary.json
|---manifest-summary.md5
|----started
```

Procedure

To migrate the DynamoDB JSON data to the Oracle NoSQL Database:

- Using a default schema
- Using a custom schema

Using a default schema

1. Prepare the configuration file (in JSON format) with the identified source and sink details. For details, see Source Configuration Templates and Sink Configuration Templates



(i) Note

If your DynamoDB exported JSON table items contain TTL attribute, to optionally import the TTL values, specify the attribute in the ttlAttributeName configuration parameter of the source configuration template and set the includeTTL configuration parameter to true in the sink configuration template.

Here, you set the defaultSchema configuration parameter to true. Therefore, the NoSQL Database Migrator creates the default schema at the sink. You must specify the DDBPartitionKey and the corresponding NoSQL column type. Otherwise, an error is displayed.

For details on the default schema for a DynamoDB exported JSON source, see *Identify the* Source and Sink topic in Workflow for Oracle NoSQL Database Migrator.

```
"source" : {
  "type" : "file",
  "format" : "dynamodb_json",
  "ttlAttributeName" : "ttl",
  "dataPath" : "<complete/path/to/the/DynamoDB/Formatted/JSON/file>"
},
"sink" : {
  "type" : "nosqldb",
  "storeName" : "kvstore",
 "helperHosts" : ["<hostname>:5000"],
  "table" : "sampledynDBImp",
  "includeTTL" : true,
  "schemaInfo" : {
    "DDBPartitionKey" : "Id:INTEGER",
    "defaultSchema" : true
 },
  "overwrite" : true,
  "requestTimeoutMs" : 5000
},
"abortOnError" : true,
"migratorVersion" : "1.7.0"
```

The following default schema is used in this example:

```
CREATE TABLE IF NOT EXISTS sampledynDBImp (Id INTEGER, document JSON,
PRIMARY KEY(SHARD(Id)))
```

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing separate configuration files for options 1 and 2. Use the --config or -c option.

```
./runMigrator --config <complete/path/to/the/JSON/config/file>
```



4. The utility proceeds with the data migration as illustrated in the following sample:

```
[INFO] creating source from given configuration:
[INFO] source creation completed
[INFO] creating sink from given configuration:
[INFO] sink creation completed
[INFO] creating migrator pipeline
[INFO] [nosqldb sink] : start loading DDLs
[INFO] [nosqldb sink] : executing DDL: CREATE TABLE IF NOT EXISTS
sampledynDBImp (Id INTEGER, document JSON, PRIMARY KEY(SHARD(Id)))
[INFO] [nosqldb sink] : completed loading DDLs
[INFO] migration started
[INFO] Start writing data to OnDB Sink
[INFO] executing for source:DynamoSample
[INFO] [DDB file source] : start parsing JSON records from file:
DynamoSample.json.qz
[INFO] Writing data to OnDB Sink completed.
[INFO] migration completed.
Records provided by source=2, Records written to sink=2, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 45ms
Migration completed.
```

Verification

Start the SQL prompt in your data store.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the new table is created with the source data:

```
SELECT * FROM sampledynDBImp
```

Output

Notice that the TTL information is included in the _metadata JSON object for each imported item.

```
{"Id":102,"document":{"Address":{"City":"Wales","DoorNum":1024,"Street":"32
main","Zip":560014},"Age":48,"FavColors":["Blue"],"FavNumbers":
[10],"FirstName":"John","LastName":"White","Phones":
[["222-222"]],"PremierCustomer":false,"_metadata":
{"expiration":1734616196000}}}
{"Id":101,"document":{"Address":{"City":"London","DoorNum":201,"Street":"21
main","Zip":570004},"Age":22,"FavColors":["Red","Green"],"FavNumbers":
[10],"FirstName":"Fred","LastName":"Smith","Phones":
[["555-222","123-567"]],"PremierCustomer":false,"_metadata":
{"expiration":1734616196000}}}
```

Using a custom schema

1. Prepare the configuration file (in JSON format) with the identified source and sink details. For details, see Source Configuration Templates and Sink Configuration Templates.



(i) Note

If your DynamoDB exported JSON table items contain TTL attribute, to optionally import the TTL values, specify the attribute in the ttlAttributeName configuration parameter of the source configuration template and set the includeTTL configuration parameter to true in the sink configuration template.

Here, you set the defaultSchema configuration parameter to false. Therefore, you specify the file containing the sink table's DDL statement in the schemaPath parameter. See Mapping of DynamoDB types to Oracle NoSQL types for more details.

The following user-defined schema is used in this example:

```
CREATE TABLE IF NOT EXISTS sampledynDBImp (Id INTEGER, document JSON,
PRIMARY KEY(SHARD(Id)))
```

NoSQL Database Migrator uses the schema file to create the table at the sink as part of the migration. As long as the primary key data is provided, the input JSON record will be inserted. Otherwise, an error is displayed.

(i) Note

- If the Dynamo DB table has a data type that is not supported in NoSQL Database, the migration fails.
- If the input data does not contain a value for a particular column (other than the primary key) then the column default value will be used. The default value must be a part of the column definition while creating the table. For example id INTEGER not null default 0. If the column does not have a default definition, SQL NULL is inserted if values are not provided for the column.
- If you are modeling DynamoDB table as a JSON document, ensure that you use AggregateFields transform in order to aggregate non-primary key data into a JSON column. For details, see aggregateFields.

```
"source" : {
   "type" : "file",
    "format" : "dynamodb_json",
    "ttlAttributeName" : "ttl",
    "dataPath" : "<complete/path/to/the/DynamoDB/Formatted/JSON/file>"
 },
  "sink" : {
    "type" : "nosqldb",
    "storeName" : "kvstore",
    "helperHosts" : ["<hostname>:5000"],
    "table" : "sampledynDBImp",
    "includeTTL" : true,
    "schemaInfo" : {
      "schemaPath" : "<full path of the schema file with the DDL
statement>"
    },
    "overwrite" : true,
```



```
"requestTimeoutMs" : 5000
},
"transforms": {
    "aggregateFields" : {
        "fieldName" : "document",
            "skipFields" : ["Id"]
      }
},
"abortOnError" : true,
"migratorVersion" : "1.7.0"
}
```

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing separate configuration files for options 1 and 2. Use the --config or -c option.

```
./runMigrator --config <complete/path/to/the/JSON/config/file>
```

4. The utility proceeds with the data migration as illustrated in the following sample:

```
[INFO] creating source from given configuration:
[INFO] source creation completed
[INFO] creating sink from given configuration:
[INFO] sink creation completed
[INFO] creating migrator pipeline
[INFO] [nosqldb sink] : start loading DDLs
[INFO] [nosqldb sink] : executing DDL: CREATE TABLE IF NOT EXISTS
sampledynDBImp (Id INTEGER, document JSON, PRIMARY KEY(SHARD(Id)))
[INFO] [nosqldb sink] : completed loading DDLs
[INFO] migration started
[INFO] Start writing data to OnDB Sink
[INFO] executing for source:DynamoSample
[INFO] [DDB file source] : start parsing JSON records from file:
DynamoSample.json.gz
[INFO] Writing data to OnDB Sink completed.
[INFO] migration completed.
Records provided by source=2, Records written to sink=2, Records
failed=0,Records skipped=0.
Elapsed time: Omin Osec 45ms
Migration completed.
```

Verification

Start the SQL prompt in your data store.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the new table is created with the source data:

```
SELECT * FROM sampledynDBImp
```

Output



Notice that the TTL information is included in the _metadata JSON object for each imported item

```
{"Id":102,"document":{"Address":{"City":"Wales","DoorNum":1024,"Street":"32
main","Zip":560014},"Age":48,"FavColors":["Blue"],"FavNumbers":
[10],"FirstName":"John","LastName":"White","Phones":
[["222-222"]],"PremierCustomer":false,"_metadata":
{"expiration":1734616196000}}}
{"Id":101,"document":{"Address":{"City":"London","DoorNum":201,"Street":"21
main","Zip":570004},"Age":22,"FavColors":["Red","Green"],"FavNumbers":
[10],"FirstName":"Fred","LastName":"Smith","Phones":
[["555-222","123-567"]],"PremierCustomer":false,"_metadata":
{"expiration":1734616196000}}}
```

Migrate from DynamoDB JSON file in AWS S3 to Oracle NoSQL Database

This example shows how to use the Oracle NoSQL Database Migrator to copy DynamoDB JSON file stored in an AWS S3 store to Oracle NoSQL Database.

Use Case:

After evaluating multiple options, an organization finalizes Oracle NoSQL Database over DynamoDB database. The organization wants to migrate their tables and data from DynamoDB to Oracle NoSQL Database.

See Mapping of DynamoDB table to Oracle NoSOL table for more details.

You can migrate a file containing the DynamoDB exported JSON data from the AWS S3 storage by specifying the path in the source configuration template.

A sample DynamoDB-formatted JSON File is as follows:

You export the DynamoDB table to AWS S3 storage as specified in <u>Exporting DynamoDB table</u> data to Amazon S3.

Example:

For this demonstration, you will learn how to migrate a DynamoDB JSON file in an AWS S3 source to Oracle NoSQL Database. You will use a manually created configuration file for this example.



Prerequisites

- Identify the source and sink for the migration.
 - Source: DynamoDB JSON File in AWS S3
 - Sink: Oracle NoSQL Database
- Identify the table in AWS DynamoDB that needs to be migrated to Oracle NoSQL
 Database. Log in to your AWS console using your credentials. Go to **DynamoDB**. Under
 Tables, choose the table to be migrated.
- Create an object bucket and export the table to S3. From your AWS console, go to S3.
 Under buckets, create a new object bucket. Go back to DynamoDB and click Exports to S3. Provide the source table and the destination S3 bucket and click Export.
 Refer to steps provided in Exporting DynamoDB table data to Amazon S3 to export your table. While exporting, you select the format as DynamoDB JSON. The exported data contains DynamoDB table data in multiple gzip files as shown below.

```
/ 01639372501551-bb4dd8c3
|-- 01639372501551-bb4dd8c3 ==> exported data prefix
|----data
|----sxz3hjr3re2dzn2ymgd2gi4iku.json.gz ==>table data
|----manifest-files.json
|----manifest-files.md5
|----manifest-summary.json
|----manifest-summary.md5
|----_started
```

 You need aws credentials (including access key ID and secret access key) and config files (credentials and optionally config) to access AWS S3 from the migrator. See <u>Set and view</u> <u>configuration settings</u> for more details on the configuration files. See <u>Creating a key pair</u> for more details on creating access keys.

Procedure

To migrate the DynamoDB JSON data to Oracle NoSQL Database, use one of the following options:

- Using a default schema
- Using a custom schema

Using a default schema

1. Prepare the configuration file (in JSON format) with the identified source and sink details. For details, see Source Configuration Templates and Sink Configuration Templates.



(i) Note

If the items in your DynamoDB JSON File in AWS S3 contain TTL attribute, to optionally import the TTL values, specify the attribute in the ttlAttributeName configuration parameter of the source configuration template and set the includeTTL configuration parameter to true in the sink configuration template. For more details, see Migrating TTL Metadata for Table Rows.

Set the defaultSchema to TRUE in the sink configuration template. The Migrator utility creates the table with the default schema at the sink. You must specify the DDBPartitionKey and the corresponding NoSQL column type. Otherwise, an error is thrown.

```
"source" : {
  "type" : "aws_s3",
  "format" : "dynamodb_json",
 "s3URL" : "<https://<bucket-name>.<s3_endpoint>/export_path>",
  "credentials" : "</path/to/aws/credentials/file>",
  "credentialsProfile" : "default"
},
"sink" : {
  "type" : "nosqldb",
  "storeName" : "kvstore",
  "helperHosts" : ["phoenix126166:5000"],
  "table" : "sampledynDBImp",
  "includeTTL" : false,
  "schemaInfo" : {
    "DDBPartitionKey" : "<PrimaryKey:Datatype>",
    "defaultSchema" : true
  "overwrite" : true,
  "requestTimeoutMs" : 5000
"abortOnError" : true,
"migratorVersion" : "1.7.0"
```

For a DynamoDB JSON source, the default schema for the table will be as shown below:

```
CREATE TABLE IF NOT EXISTS <TABLE NAME>(DDBPartitionKey name
DDBPartitionKey_type,
[DDBSortKey_name DDBSortKey_type], DOCUMENT JSON,
PRIMARY KEY(SHARD(DDBPartitionKey name),[DDBSortKey name]))
```

Where:

TABLE_NAME = value provided for the sink 'table' in the configuration

DDBPartitionKey name = value provided for the partition key in the configuration

DDBPartitionKey_type = value provided for the data type of the partition key in the configuration

DDBSortKey_name = value provided for the sort key in the configuration if any



DDBSortKey_type = value provided for the data type of the sort key in the configuration if any

DOCUMENT = All attributes except the partition and sort key of a Dynamo DB table item aggregated into a NoSQL JSON column

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing the configuration file. Use --config or -c option.

```
[~/nosqlMigrator]$./runMigrator
--config <complete/path/to/the/JSON/config/file>
```

4. The utility proceeds with the data migration, as shown below.

```
creating source from given configuration:
source creation completed
creating sink from given configuration:
sink creation completed
creating migrator pipeline
[nosqldb sink] : start loading DDLs
[nosqldb sink] : executing DDL: CREATE TABLE IF NOT EXISTS sampledynDBImp
(Id INTEGER, document JSON, PRIMARY KEY(SHARD(Id)))
[nosqldb sink] : completed loading DDLs
migration started
Start writing data to OnDB Sink
executing for source:azkzkyynnq7w7j7yqbao5377he
executing for source: jrnmvjofnu3bzlwql4cpyasuma
executing for source:kdieseglwa72xlaenwgo5vb7tg
executing for source:wp776cslg46nhm5snsi23v4j3a
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/azkzkyynnq7w7j7yqbao5377he.json.qz
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/jrnmvjofnu3bzlwq14cpyasuma.json.qz
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/kdieseqlwa72xlaenwqo5vb7tq.json.qz
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/wp776cslq46nhm5snsi23v4j3a.json.gz
[INFO] Writing data to OnDB Sink completed.
[INFO] migration completed.
Records provided by source=2, Records written to sink=2, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 747ms
Migration completed.
```

Verification

Start the SQL prompt in your KVStore.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```



Verify that the sampledynDBImp table is created with the source data:

```
{"Id":102, "document": { "Address": { "City": "Wales", "DoorNum": 1024, "Street": "32
main", "Zip":560014}, "Age":48, "FavColors": [ "Blue"], "FavNumbers":
[10], "FirstName": "John", "LastName": "White", "Phones":
[["222-222"]], "PremierCustomer": false}}
{"Id":101, "document": { "Address": { "City": "London", "DoorNum": 201, "Street": "21
main", "Zip":570004}, "Age":22, "FavColors": [ "Red", "Green"], "FavNumbers":
[10], "FirstName": "Fred", "LastName": "Smith", "Phones":
[["555-222", "123-567"]], "PremierCustomer": false}}
```

Using a custom schema

1. Prepare the configuration file (in JSON format) with the identified source and sink details. For details, see Source Configuration Templates and Sink Configuration Templates.

(i) Note

If the items in your DynamoDB JSON File in AWS S3 contain TTL attribute, to optionally import the TTL values, specify the attribute in the ttlAttributeName configuration parameter of the source configuration template and set the includeTTL configuration parameter to true in the sink configuration template. For more details, see Migrating TTL Metadata for Table Rows.

To specify a user-defined schema file in the sink configuration template, set the defaultSchema to FALSE and specify the schemaPath as a file containing your DDL statement. For details, see Mapping of DynamoDB types to Oracle NoSQL types.

(i) Note

If the Dynamo DB table has a data type that is not supported in NoSQL, the migration fails.

A sample user-defined schema file is shown below.

```
CREATE TABLE IF NOT EXISTS sampledynDBImp (AccountId INTEGER, document JSON, PRIMARY KEY(SHARD(AccountId)));
```

The schema file is used to create the DynamoDB table as fixed columns at the sink as part of the migration. As long as the primary key data is provided, the input JSON record will be inserted, otherwise it throws an error.



(i) Note

- If the input data does not contain a value for a particular column(other than the primary key) then the column default value will be used. The default value should be part of the column definition while creating the table. For example id INTEGER not null default 0. If the column does not have a default definition then SQL NULL is inserted if no values are provided for the column.
- If you are modeling DynamoDB table as a JSON document, ensure that you use AggregateFields transform in order to aggregate non-primary key data into a JSON column. For details, see aggregateFields.

```
"source" : {
    "type" : "aws_s3",
   "format": "dynamodb_json",
    "s3URL" : "<https://<bucket-name>.<s3_endpoint>/export_path>",
    "credentials" : "</path/to/aws/credentials/file>",
    "credentialsProfile" : "default"
  },
  "sink" : {
    "type" : "nosqldb",
    "storeName" : "kvstore",
    "helperHosts" : ["phoenix126166:5000"],
    "table" : "sampledynDBImp",
    "includeTTL" : false,
    "schemaInfo" : {
      "schemaPath" : "<full path of the schema file with the DDL
statement>"
    },
    "overwrite" : true,
    "requestTimeoutMs" : 5000
  "transforms" : {
    "aggregateFields" : {
      "fieldName" : "document",
      "skipFields" : ["AccountId"]
   }
  },
  "abortOnError" : true,
  "migratorVersion" : "1.7.0"
```

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing the configuration file. Use --config or -c option.

```
[~/nosqlMigrator]$./runMigrator
--config <complete/path/to/the/JSON/config/file>
```



4. The utility proceeds with the data migration, as shown below.

```
creating source from given configuration:
source creation completed
creating sink from given configuration:
sink creation completed
creating migrator pipeline
[nosqldb sink] : start loading DDLs
[nosqldb sink] : executing DDL: CREATE TABLE IF NOT EXISTS sampledynDBImp
(AccountId INTEGER, document JSON, PRIMARY KEY(SHARD(AccountId)))
[nosqldb sink] : completed loading DDLs
migration started
Start writing data to OnDB Sink
executing for source:azkzkyynng7w7j7ygbao5377he
executing for source: jrnmvjofnu3bzlwql4cpyasuma
executing for source:kdieseqlwa72xlaenwgo5vb7tq
executing for source:wp776cslq46nhm5snsi23v4j3a
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/azkzkyynnq7w7j7yqbao5377he.json.gz
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/jrnmvjofnu3bzlwql4cpyasuma.json.qz
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/kdieseqlwa72xlaenwqo5vb7tq.json.qz
[INFO] [DDB S3 source] : start parsing JSON records from object:
AWSDynamoDB/01754372757879-92f376b3/data/wp776cslq46nhm5snsi23v4j3a.json.qz
[INFO] Writing data to OnDB Sink completed.
[INFO] migration completed.
Records provided by source=2, Records written to sink=2, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 747ms
Migration completed.
```

Verification

Start the SQL prompt in your KVStore.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the sampledynDBImp table is created with the source data:



Migrate from CSV file to Oracle NoSQL Database

This example shows the usage of Oracle NoSQL Database Migrator to copy data from a CSV file to Oracle NoSQL Database.

Example

After evaluating multiple options, an organization finalizes Oracle NoSQL Database as its NoSQL Database platform. As its source contents are in CSV file format, they are looking for a way to migrate them to Oracle NoSQL Database.

In this example, you will learn to migrate the data from a CSV file called course.csv, which contains information about various courses offered by a university. You generate the configuration file from the runMigrator utility.

You can also prepare the configuration file with the identified source and sink details. See Sources and Sinks.

Prerequisites

- Identify the source and sink for the migration.
 - Source: CSV file
 In this example, the source file is course.csv

```
cat [~/nosql-migrator]/course.csv
1,"Computer Science", "San Francisco", "2500"
2,"Bio-Technology", "Los Angeles", "1200"
3,"Journalism", "Las Vegas", "1500"
4,"Telecommunication", "San Francisco", "2500"
```

- Sink: Oracle NoSQL Database
- The CSV file must conform to the RFC4180 format.
- Create a file containing the DDL commands for the schema of the target table, course. The
 table definition must match the CSV data file concerning the number of columns and their
 types.

In this example, the DDL file is mytable_schema.ddl

```
cat [~/nosql-migrator]/mytable_schema.ddl
create table course (id INTEGER, name STRING, location STRING, fees
INTEGER, PRIMARY KEY(id));
```

Procedure

To migrate the CSV file data from course.csv to Oracle NoSQL Database Service, perform the following steps:

- Open the command prompt and navigate to the directory where you extracted the Oracle NoSQL Database Migrator utility.
- 2. To generate the configuration file using Oracle NoSQL Database Migrator, execute the runMigrator command without any runtime parameters.

```
[~/nosql-migrator]$./runMigrator
```



3. As you did not provide the configuration file as a runtime parameter, the utility prompts if you want to generate the configuration now. Type y.

You can choose a location for the configuration file or retain the default location by pressing the Enter key.

```
Configuration file is not provided. Do you want to generate configuration? (y/n) [n]: y Generating a configuration file interactively.

Enter a location for your config [./migrator-config.json]: ./migrator-config.json already exist. Do you want to overwrite?(y/n) [n]: y
```

Based on the prompts from the utility, choose your options for the Source configuration.

```
Select the source:
1) nosqldb
2) nosqldb_cloud
3) file
4) object_storage_oci
5) aws_s3
#? 3

Configuration for source type=file
Select the source file format:
1) json
2) mongodb_json
3) dynamodb_json
4) csv
#? 4
```

5. Provide the path to the source CSV file. Further, based on the prompts from the utility, you can choose to reorder the column names, select the encoding method, and trim the tailing spaces from the target table.

```
Enter path to a file or directory containing csv data: [~/nosql-migrator]/course.csv

Does the CSV file contain a headerLine? (y/n) [n]: n

Do you want to reorder the column names of NoSQL table with respect to CSV file columns? (y/n) [n]: n

Provide the CSV file encoding. The supported encodings are:

UTF-8,UTF-16,US-ASCII,ISO-8859-1. [UTF-8]:

Do you want to trim the tailing spaces? (y/n) [n]: n
```

6. Based on the prompts from the utility, choose your options for the Sink configuration.

```
Select the sink:
1) nosqldb
2) nosqldb_cloud
#? 1
Configuration for sink type=nosqldb
Enter store name of the Oracle NoSQL Database: mystore
```



Enter comma separated list of host:port of Oracle NoSQL Database:
<hostname>:5000

7. Based on the prompts from the utility, provide the name of the target table.

```
Enter fully qualified table name: course
```

8. Enter your choice to set the TTL value. The default value is n.

```
Include TTL data? If you select 'yes' TTL value provided by the source will be set on imported rows. (y/n) [n]: n
```

9. Based on the prompts from the utility, specify whether or not the target table must be created through the Oracle NoSQL Database Migrator tool. If the table is already created, it is suggested to provide n. If the table is not created, the utility will request the path for the file containing the DDL commands for the schema of the target table.

```
Would you like to create table as part of migration process? Use this option if you want to create table through the migration tool. If you select yes, you will be asked to provide a file that contains table DDL or to use schema provided by the source or default schema. (y/n) [n]: y
Enter path to a file containing table DDL: [\sim/nosql-migrator]/mytable_schema.ddl
Is the store secured? (y/n) [y]: n
would you like to overwrite records which are already present?
If you select 'no' records with same primary key will be skipped [y/n] [y]: y
Enter store operation timeout in milliseconds. [5000]:
Would you like to add transformations to source data? (y/n) [n]: n
```

10. Enter your choice to determine whether to proceed with the migration in case any record fails to migrate.

```
Would you like to continue migration if any data fails to be migrated? 
 (y/n) [n]: n
```

11. The utility displays the generated configuration on the screen.

```
Generated configuration is:
{
    "source" : {
        "type" : "file",
        "format" : "csv",
        "dataPath" : "[~/nosql-migrator]/course.csv",
        "hasHeader" : false,
        "csvOptions" : {
            "encoding" : "UTF-8",
            "trim" : false
        }
}
```



```
"sink" : {
    "type" : "nosqldb",
    "storeName" : "mystore",
    "helperHosts" : ["<hostname>:5000"],
    "table" : "migrated_table",
    "query" : "",
    "includeTTL" : false,
    "schemaInfo" : {
        "schemaPath" : "[~/nosql-migrator]/mytable_schema.ddl"
      },
      "overwrite" : true,
      "requestTimeoutMs" : 5000
},
    "abortOnError" : true,
    "migratorVersion" : "1.7.0"
}
```

12. Finally, the utility prompts you to specify whether or not to proceed with the migration using the generated configuration file. The default option is y.

Note: If you select n, you can use the generated configuration file to perform the migration. Specify the ./runMigrator -c or the ./runMigrator --config option.

```
Would you like to run the migration with above configuration? If you select no, you can use the generated configuration file to run the migration using: ./runMigrator --config ./migrator-config.json (y/n) [y]: y
```

 The NoSQL Database Migrator copies your data from the CSV file to Oracle NoSQL Database.

```
creating source from given configuration:
source creation completed
creating sink from given configuration:
sink creation completed
creating migrator pipeline
migration started
[nosqldb sink] : start loading DDLs
[nosqldb sink] : executing DDL: create table course (id INTEGER, name
STRING, location STRING, fees INTEGER, PRIMARY KEY(id))
[nosqldb sink] : completed loading DDLs
[nosqldb sink] : start loading records
[csv file source] : start parsing CSV records from file: course.csv
migration completed. Records provided by source=4, Records written to
sink=4, Records failed=0, Records skipped=0.
Elapsed time: Omin Osec 559ms
Migration completed.
```

Validation



Start the SQL prompt in your KVStore.

```
java -jar lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Verify that the new table is created with the source data:

```
sql-> select * from course;
{"id":4,"name":"Telecommunication","location":"San Francisco","fees":2500}
{"id":1,"name":"Computer Science","location":"San Francisco","fees":2500}
{"id":2,"name":"Bio-Technology","location":"Los Angeles","fees":1200}
{"id":3,"name":"Journalism","location":"Las Vegas","fees":1500}
4 rows returned
```

Migrate from Oracle NoSQL Database to OCI Object Storage Using Session Token Authentication

This example shows how to use Oracle NoSQL Database Migrator with session token authentication to copy data from Oracle NoSQL Database table to a JSON file in an OCI Object Storage bucket.

Use case

As a developer, you are exploring an option to back up Oracle NoSQL Database table data to OCI Object Storage (OCI OS). You want to use session token-based authentication.

In this demonstration, you will use the OCI Command Line Interface commands (CLI) to create a session token. You will manually create a Migrator configuration file and perform data migration.

Prerequisites

- Identify the source and sink for the migration.
 - Source: users table in Oracle NoSQL Database.
 - Sink: JSON file in the OCI OS bucket
 Identify the region endpoint, namespace, bucket, and prefix for OCI OS. For the list of OCI OS service endpoints, see Object Storage Endpoints.
 - * endpoint: us-ashburn-1
 - * bucket: Migrate_oci
 - * prefix: userSession
 - * namespace: idhkvliewjzj

The namespace name for a bucket is the same as it's tenancy's namespace and is autogenerated when your tenancy is created. You can get the namespace name as follows:

- * From the Oracle NoSQL Database Cloud Service console, navigate to **Storage > Buckets**.
- * Select you **Compartment** from the **List Scope** and select the bucket. The *Bucket Details* page displays the name in **Namespace** parameter.

If you do not provide an OCI OS namespace name, the Migrator utility uses the default namespace of the tenancy.





(i) Note

Ensure that you have the privileges to write objects in the OCI OS bucket. For more details on setting the policies, see Write to Object Storage.

- Generate a session token by following these steps:
 - Install and configure OCI CLI. See Quickstart.
 - Use one of the following OCI CLI commands to generate a session token. For more details on the available options, see Token-based Authentication for the CLI.

```
#Create a session token using OCI CLI from a web browser:
oci session authenticate --region <region_name> --profile-name
cprofile_name>
```

#Example:

oci session authenticate --region us-ashburn-1 --profile-name SESSIONPROFILE

or

#Create a session token using OCI CLI without a web browser: oci session authenticate --no-browser --region <region_name> --profilename <profile name>

#Example:

oci session authenticate --no-browser --region us-ashburn-1 --profilename SESSIONPROFILE

In the command above.

region_name: Specifies the region endpoint for your OCI OS. For a list of data regions supported in Oracle NoSQL Database Cloud Service, see Data Regions and Associated Service URLs.

profile_name: Specifies the profile, which the OCI CLI command uses to generate a session token.

The OCI CLI command creates an entry in the OCI config file at \$HOME / .oci / config path as shown in the following sample:

[SESSIONPROFILE]

```
fingerprint=f1:e9:b7:e6:25:ff:fe:05:71:be:e8:aa:cc:3d:0d:23
key_file=$HOME/.oci/sessions/SESSIONPROFILE/oci_api_key.pem
tenancy=ocid1.tenancy.oc1..aaaaa ... d6zjq
region=us-ashburn-1
security_token_file=$HOME/.oci/sessions/SESSIONPROFILE/token
```

The security_token_file points to the path of the session token that you generated using the OCI CLI command above.



Note

- * If the profile already exists in the OCI config file, the OCI CLI command overwrites the profile with session-token related configuration while generating the session token.
- * Specify the following in your sink configuration template:
 - * The path to the OCI config file in the credentials parameter.
 - * The profile used while generating the session token in the credentialsProfile parameter.

```
"credentials" : "$HOME/.oci/config"
"credentialsProfile" : "SESSIONPROFILE"
```

The Migrator utility automatically fetches the details of the session token generated using the parameters above. If you don't specify the credentials parameter, the Migrator utility looks for the credentials file in the path \$HOME/.oci. If you don't specify the credentialsProfile parameter, the Migrator utility uses the default profile name (DEFAULT) from the OCI config file.

* The session token is valid for 60 minutes. To extend the session duration, you can refresh the session. For details, see Refreshing a Token.

Procedure

To migrate from Oracle NoSQL Database table to a JSON file in the OCI OS bucket:

1. Prepare the configuration file (in JSON format) with Oracle NoSQL Database source and JSON file in the OCI OS bucket sink. For templates, see Source Configuration Templates and Sink Configuration Templates.

To use the session token authentication to access OCI OS bucket, set the useSessionToken parameter to true in the sink configuration template. Correspondingly, specify the config path in the credentials parameter and the profile name in the credentialsProfile parameter.

```
"source" : {
  "type" : "nosqldb",
  "storeName" : "kvstore",
  "helperHosts" : ["<hostname>:<port>"],
  "table" : "users",
  "includeTTL" : true,
  "requestTimeoutMs" : 5000
},
"sink" : {
  "type" : "object_storage_oci",
  "format" : "json",
  "endpoint" : "us-ashburn-1",
  "namespace" : "idhkvliewjzj",
  "bucket" : "Migrate oci",
  "prefix" : "userSession",
  "chunkSize" : 32,
  "compression" : "",
  "useSessionToken" : true,
```



```
"credentials" : "$/home/.oci/config",
    "credentialsProfile" : "SESSIONPROFILE"
},
"abortOnError" : true,
"migratorVersion" : "1.7.0"
}
```

- Open the command prompt and navigate to the directory where you extracted the NoSQL Database Migrator utility.
- 3. Run the runMigrator command by passing configuration file option. Use the --config or -c option to pass the configuration file as follows:

```
./runMigrator --config ./migrator-config.json
```

4. The Migrator utility proceeds with data migration. A sample output is shown below.

With useSessionToken parameter to true, the Migrator utility automatically authenticates using the session token. The Migrator utility copies your data from users table to a JSON file in the OCI OS bucket named Migrate_oci. Check the logs for successful data backup.

```
[INFO] creating source from given configuration:
[INFO] source creation completed
[INFO] creating sink from given configuration:
[INFO] sink creation completed
[INFO] creating migrator pipeline
[INFO] [OCI OS sink] : writing table schema to userSession/Schema/
schema.ddl
[INFO] migration started
[INFO] Migration success for source users 6 10. read=2, written=2, failed=0
[INFO] Migration success for source users 1 5. read=3, written=3, failed=0
[INFO] Migration is successful for all the sources.
[INFO] migration completed.
Records provided by source=5, Records written to sink=5, Records
failed=0, Records skipped=0.
Elapsed time: Omin Osec 982ms
Migration completed.
```

(i) Note

Depending on the chunkSize parameter in the sink configuration template, the Migrator utility splits the source data into several JSON files in the same directory. In this example, the Migrator utility copies data to users_1_5_0.json and users_6_10_0.json files in Migrate_oci/userSession/Data directory.

The source table schema is copied to schema.ddl file in Migrate_oci/userSession/Schema directory.

Verification

To verify your data backup, log in to the Oracle NoSQL Database Cloud Service console. Navigate through the menus, Storage > Object Storage & Archive Storage > Buckets. Access the files from the userSession directory in the Migrate_oci bucket. For the procedure to access the console, see Accessing the Service from the Infrastructure Console



Troubleshooting the Oracle NoSQL Database Migrator

Learn about the general challenges that you may face while using the , and how to resolve them.

Migration has failed. How can I resolve this?

A failure of the data migration can be because of multiple underlying reasons. The important causes are listed below:

Table 1-5 Migration Failure Causes

Error Message	Meaning	Resolution
Failed to connect to Oracle NoSQL Database	The migrator could not establish a connection with the NoSQL Database.	 Check if the values of the storeName and helperHosts attributes in the configuration JSON file are valid and that the hosts are reachable. For a secured store, verify if the security file is valid with correct user name and password values.
Failed to connect to Oracle NoSQL Database Cloud Service	The migrator could not establish a connection with the Oracle NoSQL Database Cloud Service.	· · · · · · · · · · · · · · · · · · ·



Table 1-5 (Cont.) Migration Failure Causes

Error Message	Meaning	Resolution
Table not found	The table identified for the migration could not be located by the NoSQL Database Migrator.	 Verify if the table is present in the source database. Ensure that the table is qualified with its namespace in the configuration JSON file, if the table is created in a non-default namespace. Verify if you have the required read/write authorization to access the table. If the source is Oracle NoSQL Database Cloud Service, verify if the valid compartment name is specified in the configuration JSON file, and ensure that you have the required authorization to access the table.
		 Verify if the table is present in the Sink. If it does not exist, you must either create the table manually or use the schemaInfo config to create it through the migration.
DDL Execution failed	The DDL commands provided in the input schema definition file is invalid.	 Check the syntax of the DDL commands in the schemaPath file. Ensure that there is only one DDL statement per line in the schemaPath file.
failed to write record to the sink table with java.lang.IllegalArgumentE xception	The input record is not matching with the table schema of the sink.	 Check if the data types and column names specified in the target sink table are matching with sink table schema. If you applied any transformation, check if the transformed records are matching with the sink table schema.
Request timeout	The source or sink's operation did not complete within the expected time.	 Verify the network connection. Check if the NoSQL Database is up and running. Try to increase requestTimeout value in the configuration JSON file.



What should I consider before restarting a failed migration?

When a data migration task fails, the sink will be at an intermediate state containing the imported data until the point of failure. You can identify the error and failure details from the logs and restart the migration after diagnosing and correcting the error. A restarted migration starts over, processing all data from the beginning. There is no way to checkpoint and restart the migration from the point of failure. Therefore, NoSQL Database Migrator overwrites any record that was migrated to the sink already.

Best Practices

The time taken for the data migration depends on multiple factors such as volume of data being migrated, network speed, current load on the database. In case of a cloud service, the speed of migration also depends on the read throughput and the write throughput provisioned. So, to improve the migration speed, you can:

- Consider running the migration during off-hours when the load on the database is less.
- Consider allocating the VM where the NoSOL Database Migrator will run, defining the data source, and defining the data sink in the same OCI region to ensure minimal network latencies.
- In case of Oracle NoSQL Database Cloud Service, verify if the storage allocated for table is sufficient. If the NoSQL Database Migrator is not creating the table, you can increase the write throughput. If the migrator is creating the table, consider specifying a higher value for the schemaInfo.writeUnits parameter in the sink configuration. After the data migration completes, you can lower this value.



(i) Note

There is no limitation on the number of times you can increase the throughput or storage limits. You can decrease the throughput or storage limits only up to 4 times in a 24-hour period. see Cloud Limits and Sink Configuration Templates .

The Migrator utility is inherently designed to achieve higher migration speed by processing multiple streams in parallel. The following points suggest how to leverage this capability for various migration scenarios:

Migrating from Oracle NoSQL Database Cloud Service/on-premises tables to File system/Object Storage sink:

Set useMultiFiles and chunkSize parameters in the Migrator configuration. The useMultiFiles parameter creates multiple files/objects at the sink. The chunkSize parameter determines the size of each file during data export.

For example: To export 2 GB data, setting the useMultiFiles parameter to true and chunkSize parameter to 40MB causes the Migrator utility to write 50 files of 40 MB each.



(i) Note

The Migrator utility can currently process 100 streams in parallel. Therefore, set the chunkSize parameter to an optimal file size value such that the Migrator utility creates a maximum of 100 files during data export.

Migrating from a File system/Object Storage to Oracle NoSQL Database Cloud Service/on-premises sink:



- If your File system/Object Storage has exported data containing multiple files/objects from a previous migration, the Migrator utility automatically processes files in parallel to achieve higher migration speed while importing the data.
- If you are migrating data from other external File systems/Object Storage, consider splitting data into multiple files/multiple objects at the data source.

Note

- In case of Oracle NoSQL Database Cloud Service sink, you must configure sufficient write throughput and table write units percentage to process up to 100 streams during the migration operation.
- If you have more than 100 source files, the Migrator utility creates a maximum of 100 streams and distributes the files among them during data import. The files in each stream will be sequentially migrated.

I have a long running migration involving huge datasets. How can I track the progress of the migration?

You can enable additional logging to track the progress of a long-running migration. To control the logging behavior of Oracle NoSQL Database Migrator, you must set the desired level of logging in the logging.properties file. This file is provided with the NoSQL Database Migrator package and available in the directory where the Oracle NoSQL Database Migrator was unpacked. The different levels of logging are OFF, SEVERE, WARNING, INFO, FINE, and ALL in the order of increasing verbosity. Setting the log level to OFF turns off all the logging information, whereas setting the log level to ALL provides the full log information. The default log level is WARNING. All the logging output is configured to go to the console by default. You can see comments in the logging.properties file to know about each log level.

Glossary

Index