ONEHOUSE

Bring Your Own Kafka for SQL Server CDC with Onehouse

Contents

Introduction

Change data capture (CDC) is a methodology in data management that enables the real-time replication of data across different systems. A common use case for CDC is to keep a downstream analytics database, such as a data lakehouse, in sync with an operational database.

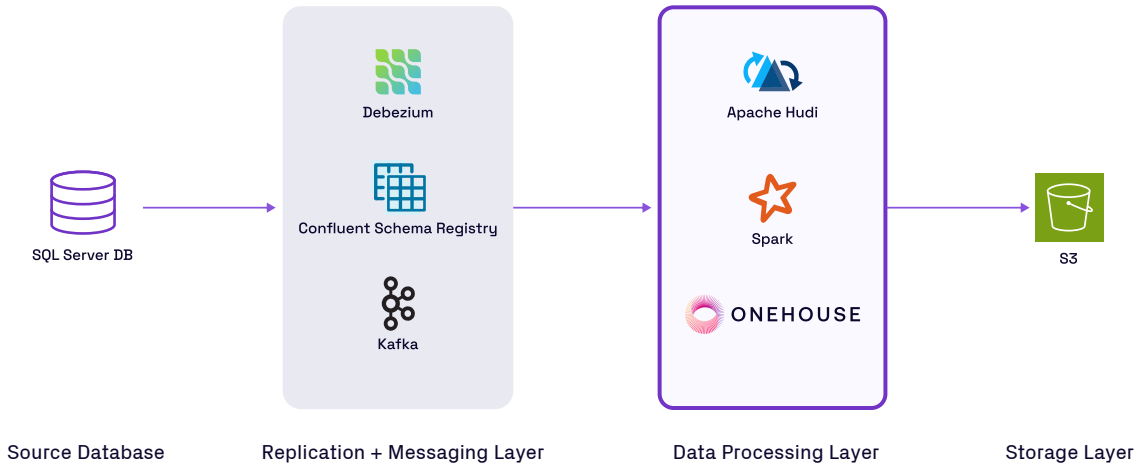
Onehouse offers end-to-end replication for database sources such as:

1. PostgreSQL and MySQL, with direct support, including both on-premises and cloud-based deployments.
2. SQL Server, using bring your own Kafka implementations.

In this guide, we'll look into one of the ways to implement a fully-managed CDC from a SQL Server database to a data lakehouse using Confluent Cloud's managed Kafka Connect, Confluent Schema Registry, and Onehouse. Optionally, you can use Apache XTable (Incubating) to also use Apache Iceberg-formatted files and/or Delta Lake-formatted files.

Note: This guide describes using Onehouse Cloud in a BYOC (Bring Your Own Cloud) deployment model. Onehouse Cloud is a managed service that handles all the ugly details for you. If you are an open-source Hudi user, consider Onehouse LakeView (free) and Onehouse Table Optimizer (managed service), each of which handle some of the ugly details for you.

Architecture Walkthrough



Debezium

Debezium offers a set of distributed services that capture row-level changes in your database, so your applications can see and respond to those changes. Debezium records all row-level changes committed to each database table in a transaction log.

Apache Kafka

Kafka, a powerful distributed event streaming platform, plays a crucial role in implementing CDC, by efficiently handling high-throughput data streams. In a CDC architecture, Debezium and Apache Kafka are coupled; Debezium captures database row-level changes as events and publishes them to Kafka topics.

Confluent Cloud

Confluent Cloud is a fully managed Kafka service, further simplifying streaming by offering a scalable and reliable infrastructure for real-time data integration. In this architecture, Confluent Cloud manages Debezium, Apache Kafka, and Schema Registry deployments.

Onehouse

Onehouse is a fully managed Universal Data Lakehouse platform that deploys and manages data infrastructure components, enabling full automation of streaming pipelines that deliver data from your source systems to your target applications. With Onehouse, you can easily ingest and transform data from any source, manage it centrally in a data lakehouse, and query or access it with the engine and table format of your choice.

In this architecture, Onehouse manages provisioning infrastructure required for data processing, which includes Apache Hudi and Apache Spark.

Together, these technologies empower organizations to seamlessly capture, process, and analyze data changes, enhancing their ability to make data-driven decisions and maintain data consistency across various applications and services.

Steps

SQL Server

In your SQL Server database, run the following command to enable CDC for the current database. This procedure must be executed for a database before any tables can be enabled for CDC in that database.

```
USE databaseName;

EXEC sys.sp_cdc_enable_db;
GO
```

Next, enable CDC for the specified source table in the current database. When a table is enabled for CDC, a record of each data manipulation language (DML) operation applied to the table is written to the transaction log. The CDC process retrieves this information from the log and writes it to change tables that are accessed by using a set of functions.

```
EXEC sys.sp_cdc_enable_table @source_schema = 'dbo', @source_name = 'tableName',
@role_name = NULL, @supports_net_changes = 0;
GO
```

Confluent Cloud

Deploying Connector

To deploy the **Microsoft SQL Server CDC Source V2 (Debezium) Connector** in Confluent Cloud, follow the steps provided in Confluent's documentation. The V2 Connector automatically creates a topic which can be directly consumed by target applications such as Onehouse.

Adding Schema to Schema Registry

Let's say your source table, products, uses a schema such as the one below in your SQL Server database.

```
{
"name": "id",
"type": "long"
},
{
"name": "name",
"type": "string"
},
{
"name": "quantity",
"type": "long"
}
```

Create a schema named **`productSchema`** in **`Confluent Schema Registry`** by following this documentation with the schema below.

```
{
  "fields": [
    {
      "default": null,
      "name": "after",
      "type": [
        "null",
        {
          "fields": [
            {
              "name": "id",
              "type": "long"
            },
            {
             "name": "name",
              "type": "string"
            },
            {
              "name": "quantity",
              "type": "long"
            }
          ],
```

```
        "name": "After",
        "type": "record"
      }
    ]
  },
  {
    "default": null,
    "name": "before",
    "type": [
      "null",
      {
        "fields": [
          {
            "name": "id",
            "type": "long"
          },
          {
            "name": "name",
            "type": "string"
          },
          {
            "name": "quantity",
            "type": "long"
          }
        ],
        "name": "Before",
        "type": "record"
      }
    ]
  },
  {
    "name": "op",
    "type": "string"
  },
  {
    "name": "source",
    "type": {
      "fields": [
        {
          "name": "change_lsn",
          "type": "string"
        },
        {
          "name": "commit_lsn",
          "type": "string"
        },
        {
          "name": "connector",
          "type": "string"
        },
        {
          "name": "db",
          "type": "string"
        },
```

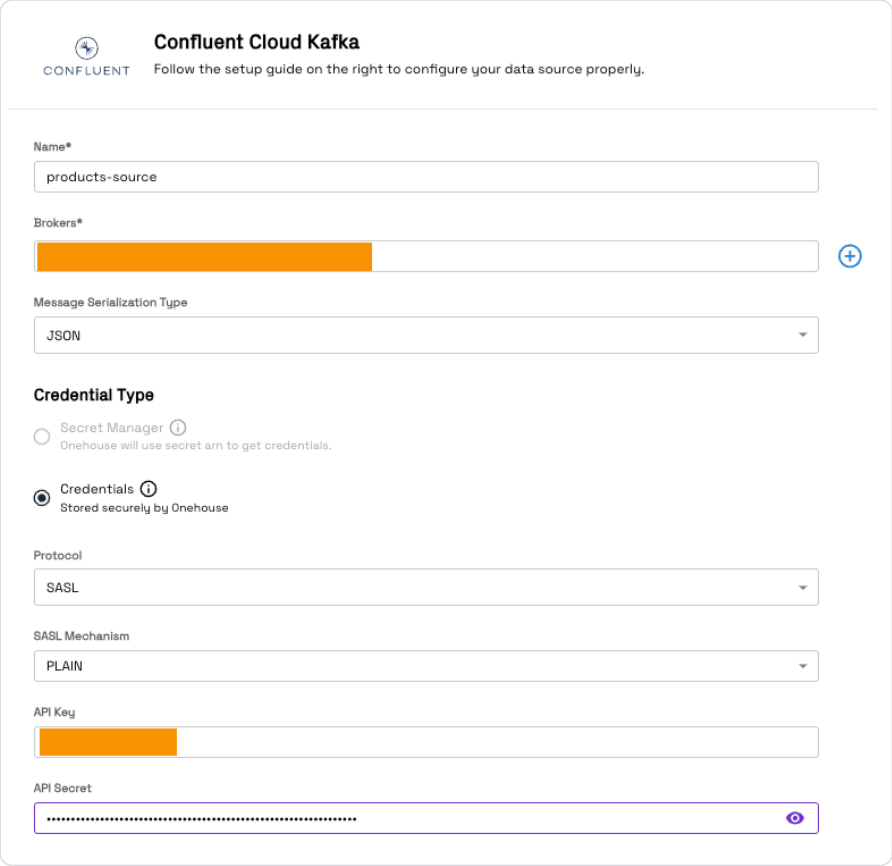
```
        {
          "name": "event_serial_no",
          "type": "string"
        },
        {
          "name": "name",
          "type": "string"
        },
        {
          "name": "schema",
          "type": "string"
        },
        {
          "name": "sequence",
          "type": "string"
        },
        {
          "name": "snapshot",
          "type": "string"
        },
        {
          "name": "table",
          "type": "string"
        },
        {
          "name": "ts_ms",
          "type": "long"
        },
        {
          "name": "version",
          "type": "string"
        },
        {
          "name": "transaction",
          "type": "string"
        }
      ],
      "name": "Source",
      "type": "record"
    }
  },
  {
    "name": "ts_ms",
    "type": "long"
  },
  {
    "name": "schema",
    "type": {
      "fields": [
        {
          "name": "fields",
          "type": {
            "items": {
              "fields": [
                {
                  "name": "name",
                  "type": "string"
                },
```

```
                {
                  "name": "optional",
                  "type": "boolean"
                },
                {
                  "name": "type",
                  "type": "string"
                },
                {
                  "name": "version",
                  "type": "long"
                }
              ],
              "name": "Field",
              "type": "record"
            },
            "type": "array"
          }
        }
      ],
      "name": "Schema",
      "type": "record"
    }
  }
 ],
 "name": "Payload",
 "type": "record"
}
```

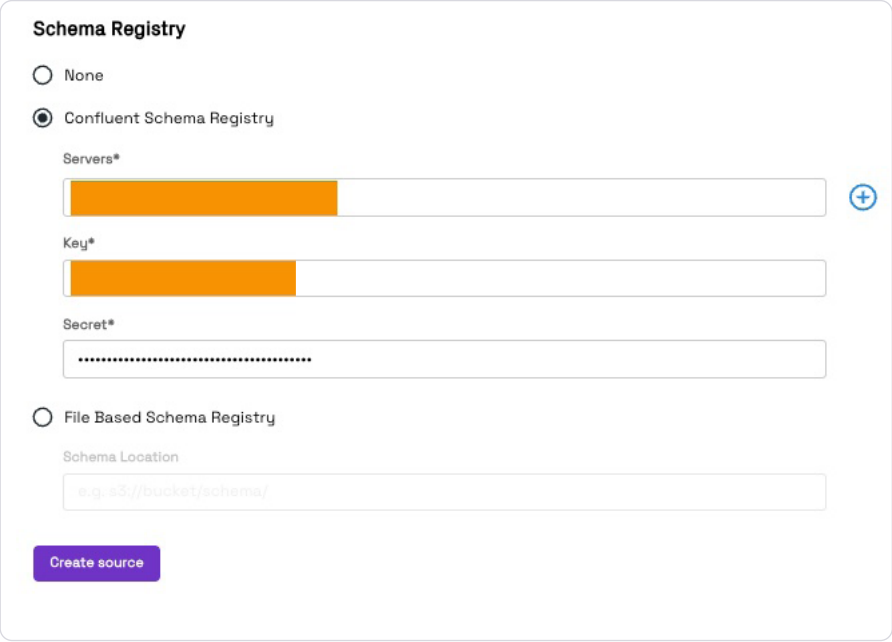
Onehouse

Create a Source

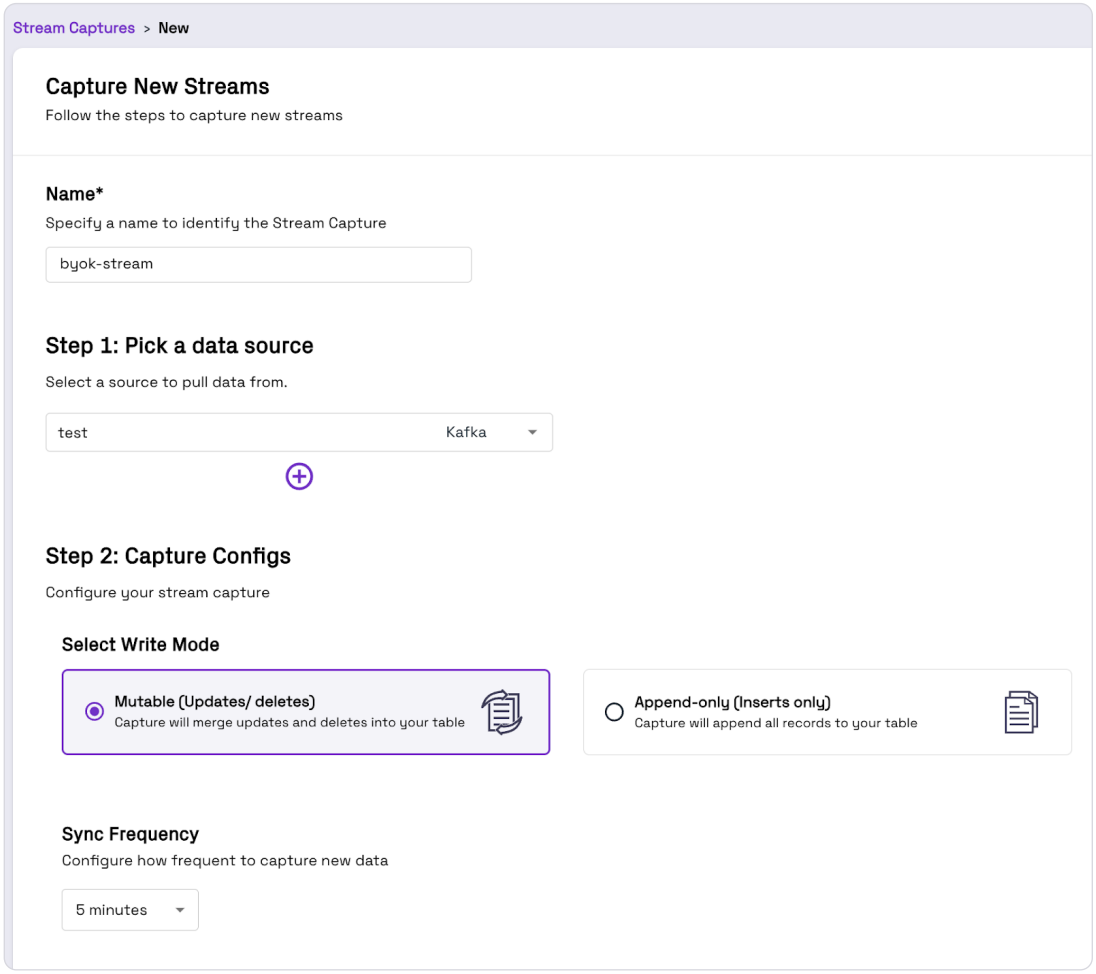
Create a `Confluent Cloud Kafka` source in Onehouse by adding your `Broker` endpoint URL, `API Key` and `API Secret`



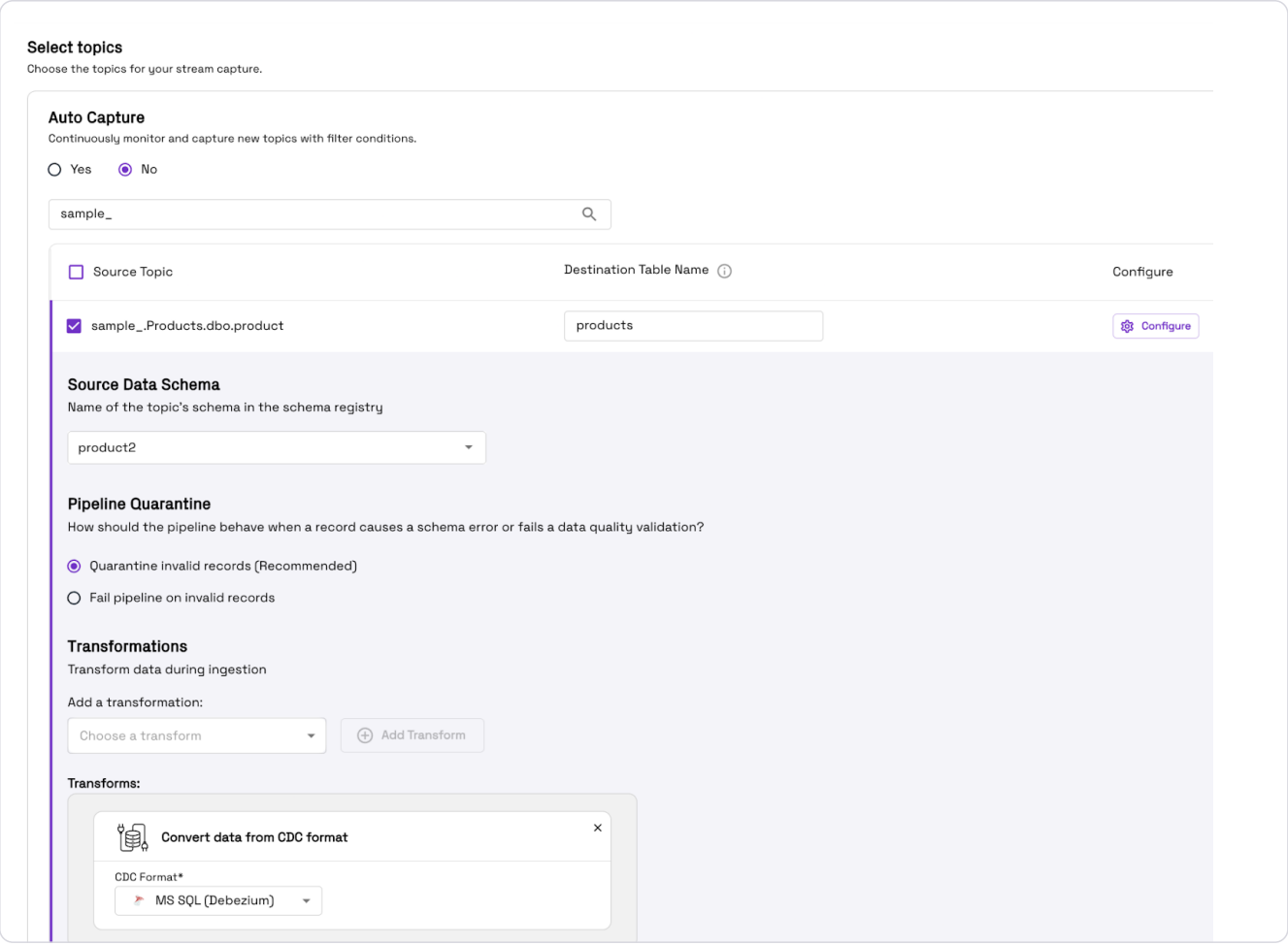
In the same screen, provide your Schema Registry `Server endpoint`, `Key` and `Secret` values, and then click `Create source`.



- From the Stream Captures screen, pick the appropriate `Name` and `Sync Frequency` while selecting the right source, i.e. **`products-source`**.
- Also select the desired write mode. For this CDC example, we expect updates to the source database to be propagated to the target table, so we select **Mutable** as the write mode. (Read more about **Mutable** vs **Append-only** write mode here.)



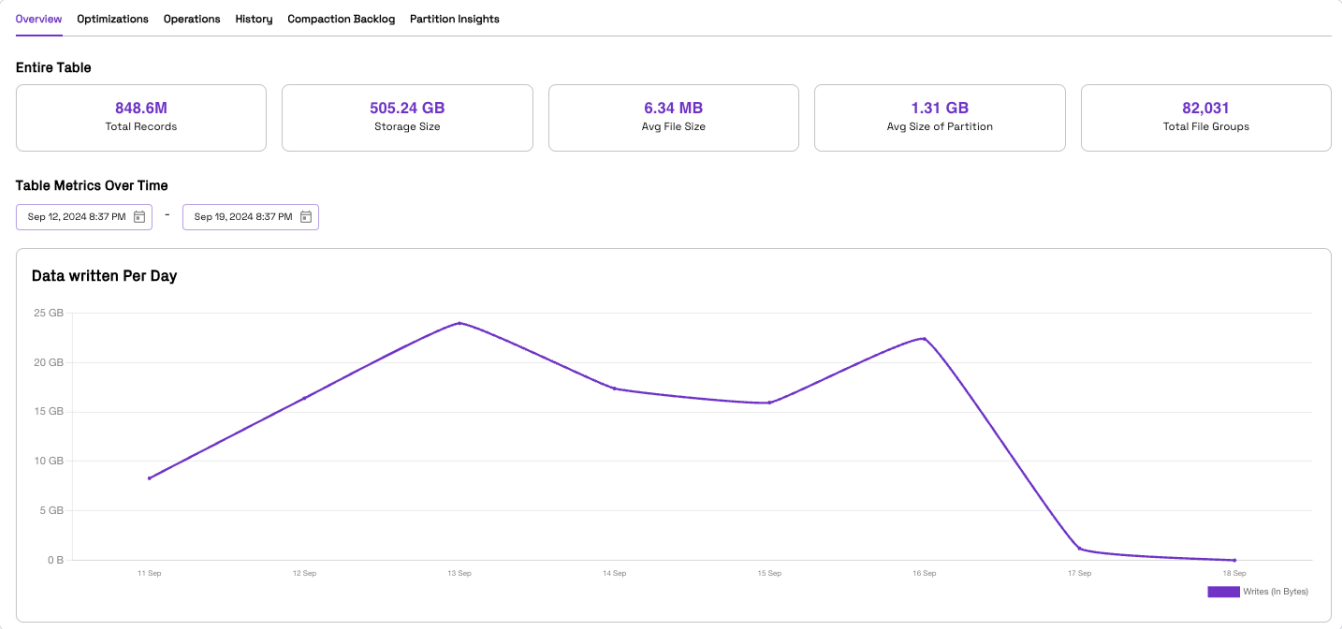
- Next, select the right schema name, i.e. **productSchema**, in the Source Data Schema field, then choose **Convert data from CDC format** in the `Add a transformation` field.
- Next, choose appropriate `Data Quality Validation` and `Starting Offsets` while selecting the id column as the `record key` and the **`commit_lsn`** column as the `precombine field`, assuming you're working with a schema as highlighted in the `Adding Schema to Schema Registry` section.

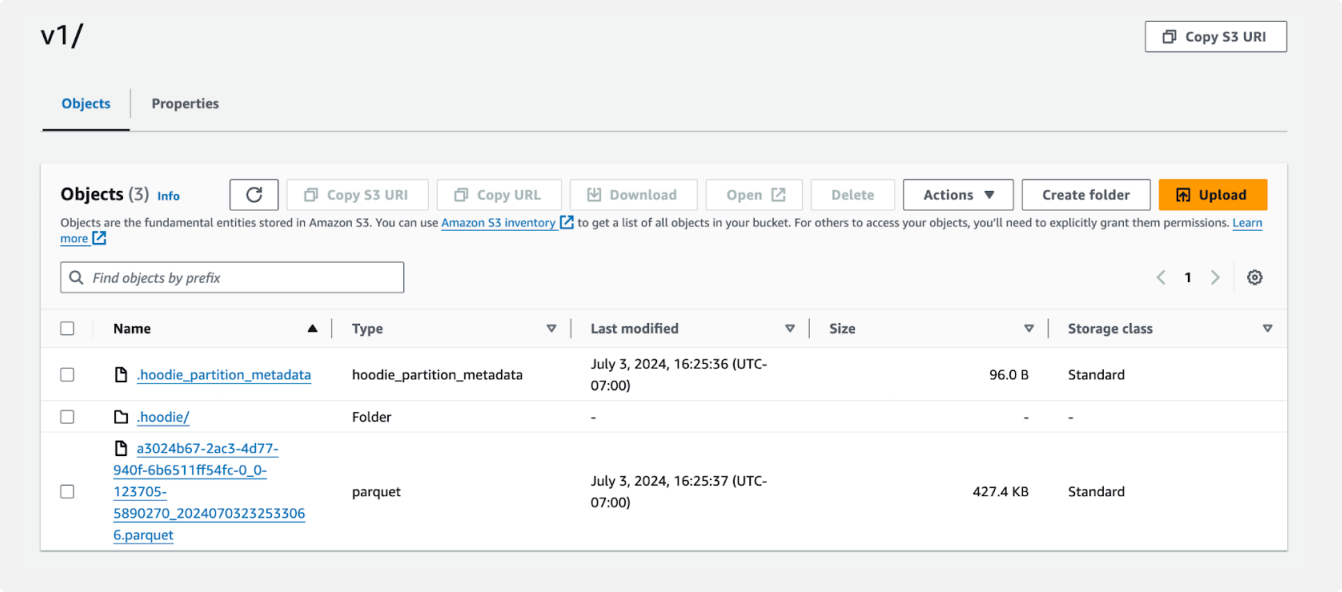


- Then proceed to select the appropriate `data lake`, `database,` and `Catalog`, and select `Create Stream Capture`.
- If your goal is to query the target table in Delta Lake or Apache Iceberg table formats, you can create an Apache XTable catalog by following the instructions here. As Onehouse can do multi-catalog synchronization simultaneously, all your warehouses and query engines can query the tables managed by a single pipeline.

Validation

Once your pipeline starts running, you'll be able to see the records populated in your configured data lake.





Conclusion

In this guide, we have implemented an end-to-end CDC architecture to capture changes from your source SQL Server database through Debezium with Confluent Cloud Kafka, and created a streaming pipeline with Onehouse to create a fully interoperable data lakehouse.

If you want to learn more about Onehouse and would like to give it a try, please visit the Onehouse listing on the AWS Marketplace or contact gtm@onehouse.ai.