ONEHOUSE

Getting Started Guide

Synchronize PostgreSQL and your Lakehouse: CDC with Onehouse on AWS

Edition: January 2nd, 2024

Introduction

Organizations have a wealth of data sitting in their business applications that can be mined for insights and decision making. Most of this application data is hosted within various OLTP databases.

The traditional approach to analytics on business data is to build a data warehouse, partitioned and organized to serve specific types of queries. However, this requires an understanding, in advance, of the kinds of questions the business will have, and it fails to support workloads in data science, such as predictive analytics and AI/ML.

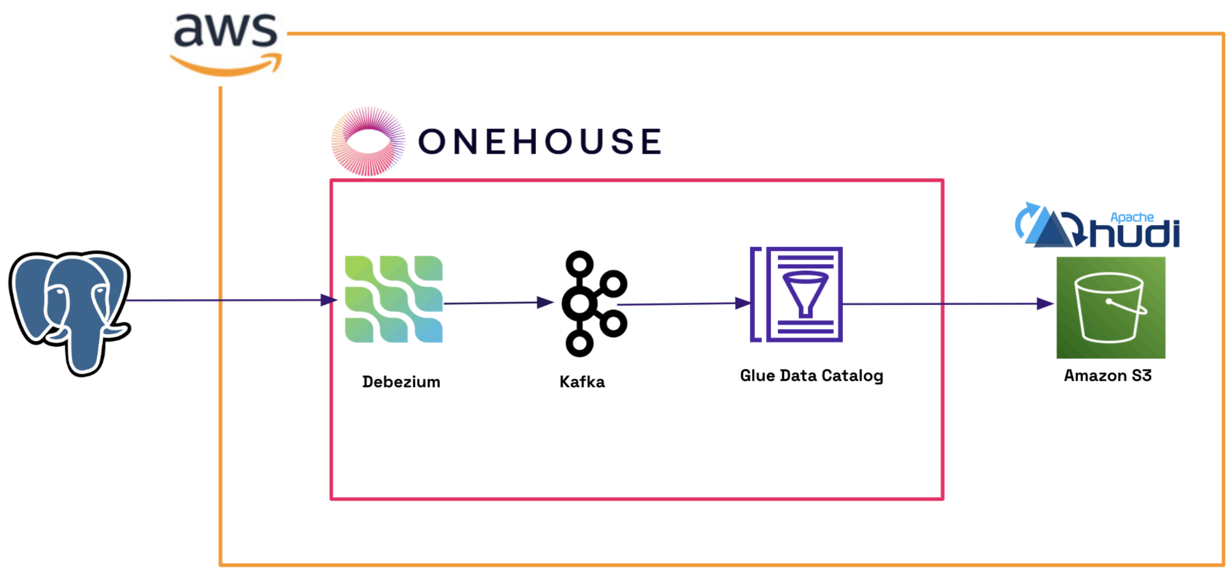
Another solution that is gaining popularity is replicating this data in near real-time to a data lakehouse. This requires less upfront planning, is more flexible, is less expensive to create and operate, and is capable of serving a wide variety of use cases.

In this guide, we will show you how to set up the Change Data Capture (CDC) feature of Onehouse to enable continuous synchronization of data from an OLTP database into a data lakehouse. We will be using Amazon RDS PostgreSQL as our example source database.

Solution Overview

Onehouse can automatically configure an end-to-end CDC ingestion pipeline from a PostgreSQL database (with other RDBMS's on the roadmap) to an analytics-ready data lakehouse.

Several Onehouse customers, such as Apna, have been running this pipeline in production. Under the hood, Onehouse automates the data infrastructure in your account to leverage Apache Kafka, Debezium, Apache Spark, and Apache Hudi, without exposing any of this complexity or required maintenance to you. If you are currently designing your own CDC ingestion pipelines to a data lake, or if you are tired of the maintenance requirements, inflexibility and on-call monitoring of an in-house solution, Onehouse can make this a seamless experience for you.

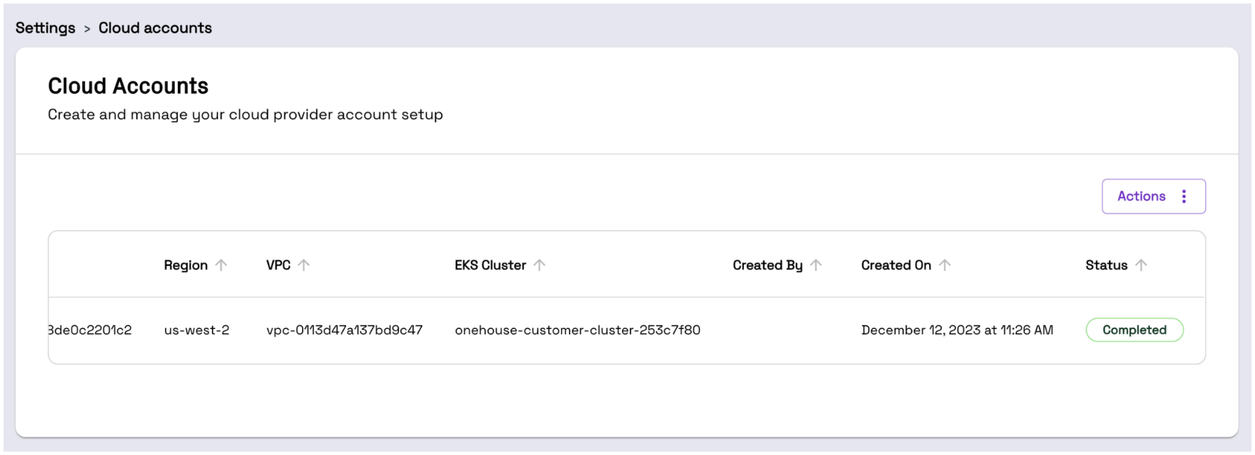


We will set up Onehouse to ingest data continuously from an AWS RDS PostgreSQL database into an Amazon S3 data lake.

Prerequisites

- A fully linked and provisioned Onehouse project on AWS

  To check this, in Onehouse navigate to `Settings -> Cloud accounts` and check that you have an entry in the table with a status of 'Completed'.



Settings > Cloud accounts

Cloud Accounts

Create and manage your cloud provider account setup

	Region	VPC	EKS Cluster	Created By	Created On	Status
3de0c2201c2	us-west-2	vpc-0113d47a137bd9c47	onehouse-customer-cluster-253c7f80		December 12, 2023 at 11:26 AM	Completed

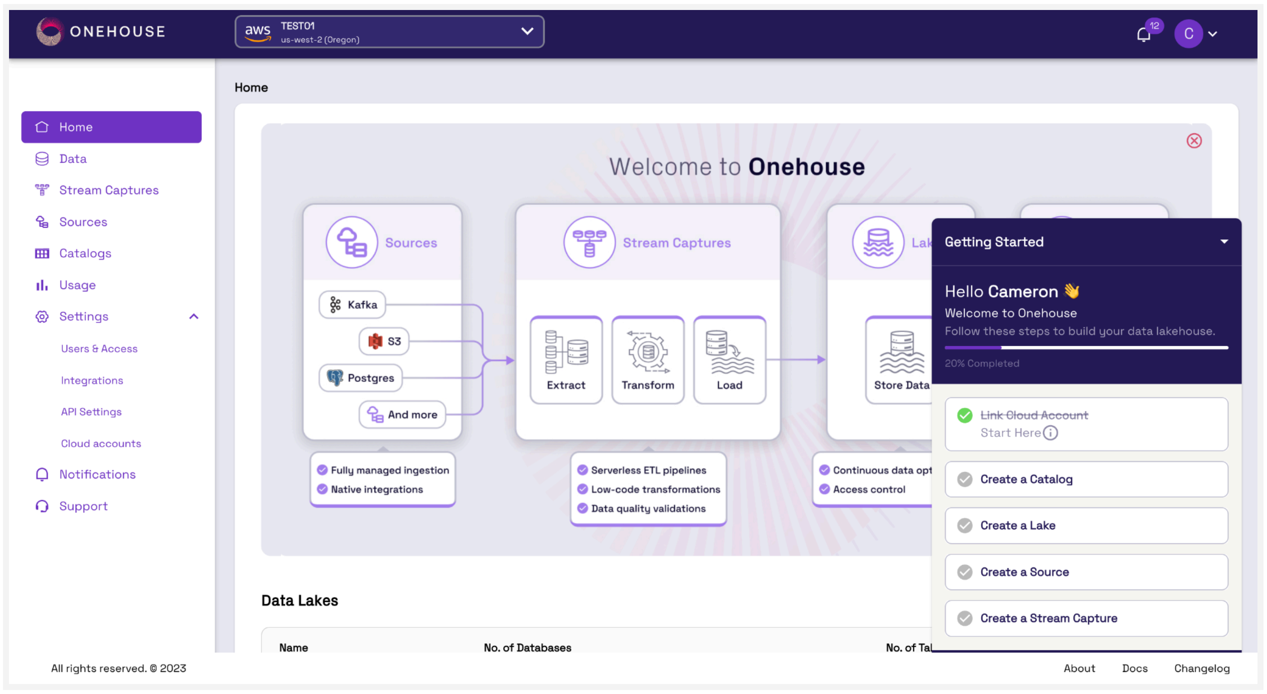
If you haven't yet provisioned your Onehouse project, please refer to this guide:

- (Onehouse provisioning guide coming soon…)

If you still need a login to Onehouse, please sign up for a free test drive at: https://www.onehouse.ai/schedule-a-test-drive

Additional Onehouse Setup

Before setting up any data ingestion, we first need to create a data lake and at least one data catalog. If the homepage of your Onehouse account shows that you still need to 'Create a Catalog' and 'Create a Lake' in the 'Getting Started' dialog, then you are in the right place. If you already have a data lake and a catalog set up, you can skip to the next step.



Getting Started

Hello Cameron 👋

Welcome to Onehouse

Follow these steps to build your data lakehouse.

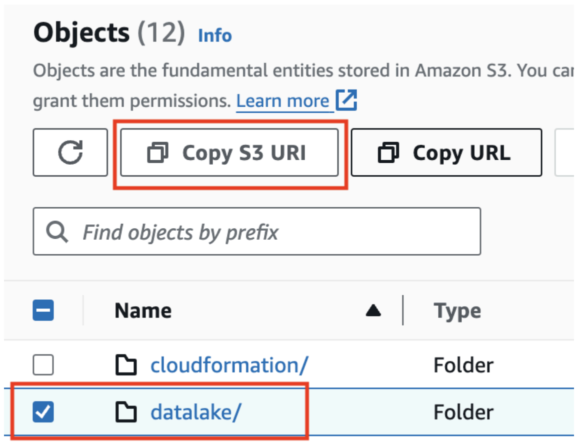
20% Completed

- ✓ ~~Link Cloud Account~~ Start Here
- Create a Catalog
- Create a Lake
- Create a Source
- Create a Stream Capture

Create a data lake

- ☐ Create a folder for the data lake inside the Onehouse AWS S3 bucket.
    - ☐ Log in to your AWS console, and navigate to the 'Amazon S3' service and then 'Buckets'. You should see a bucket name beginning with `onehouse-customer-bucket`.
    - ☐ Click the bucket name to see the Objects inside the bucket, then click 'Create folder'. Name the new folder 'datalake' and click the 'Create folder' button.
    - ☐ While you are here, create an additional folder called 'query_results'. We will use this folder later.
- ☐ Register the data lake in Onehouse
    - ☐ On the Onehouse portal, navigate to the 'Data' tab, then click the 'Create Lake' button.
    - ☐ Give the new data lake a name. We'll call it `acme_retail`.
    - ☐ For 'Type' click 'Managed Lake'

- [ ] For 'Root Path' copy the URI of the S3 bucket and folder using the 'Copy S3 URI' button in Amazon S3.

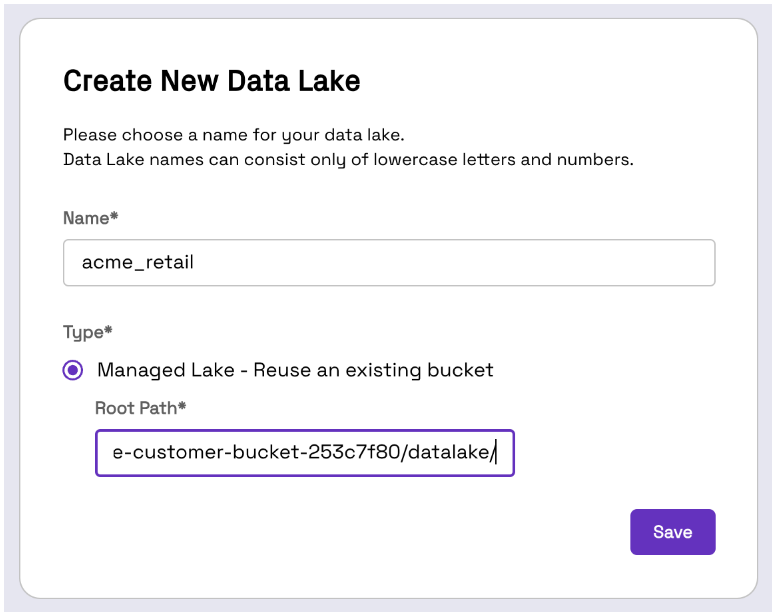


Objects (12) Info

Objects are the fundamental entities stored in Amazon S3. You ca
grant them permissions. Learn more

	Name	Type
☐	cloudformation/	Folder
☑	datalake/	Folder

- [ ] Back in Onehouse, paste the S3 URI into the 'Root Path' field, then click 'Save'.



Create New Data Lake

Please choose a name for your data lake.
Data Lake names can consist only of lowercase letters and numbers.

Name*: acme_retail

Type*: ● Managed Lake - Reuse an existing bucket

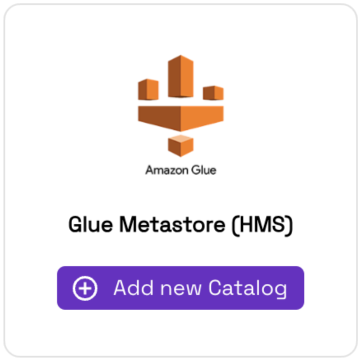
Root Path*: e-customer-bucket-253c7f80/datalake/

Create a database

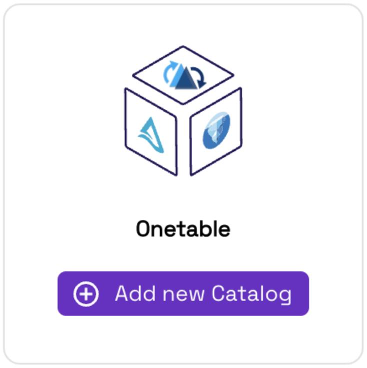
- Name it 'acme_retail_bronze'
- You can delete the database called 'acme_retail_default'

Create data catalogs

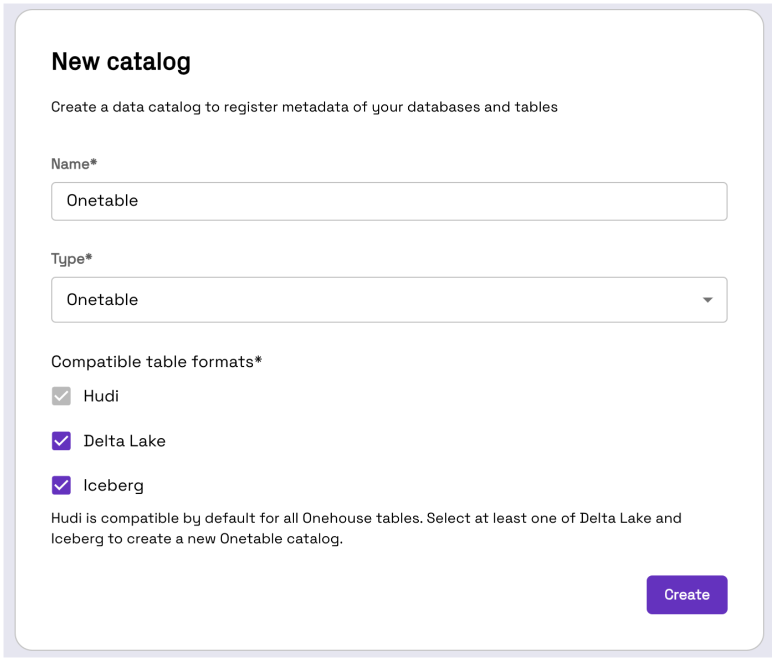
- ☐ In Onehouse, navigate to the ‘Catalogs’ tab.
- ☐ In the ‘Glue Metastore’ card, click ‘Add new Catalog’.



- ☐ Give the new catalog a name like ‘Onehouse’ and click the ‘Create’ button.
- ☐ In the ‘Onetable’ card, click ‘Add new Catalog’.



- [ ] Give the new catalog a name, such as ‘Onetable’. Under ‘Compatible table formats’ make sure that ‘Hudi’, ‘Delta Lake’ and ‘Iceberg’ are all checked.



New catalog

Create a data catalog to register metadata of your databases and tables

Name*: Onetable

Type*: Onetable

Compatible table formats*

- [x] Hudi
- [x] Delta Lake
- [x] Iceberg

Hudi is compatible by default for all Onehouse tables. Select at least one of Delta Lake and Iceberg to create a new Onetable catalog.

Note: Each additional table format you enable requires a relatively small amount of additional storage (for each format’s metadata), a slight increment in compute required, and a small performance penalty. There is no penalty on reads from the enabled table formats.

With all three table formats chosen, Onetable will write metadata in all of them. This enables readers using any of the three formats to access your data lakehouse with no performance penalty. We will use this later to demonstrate connectivity from various query engines.

Create a Postgres Database (Amazon RDS)

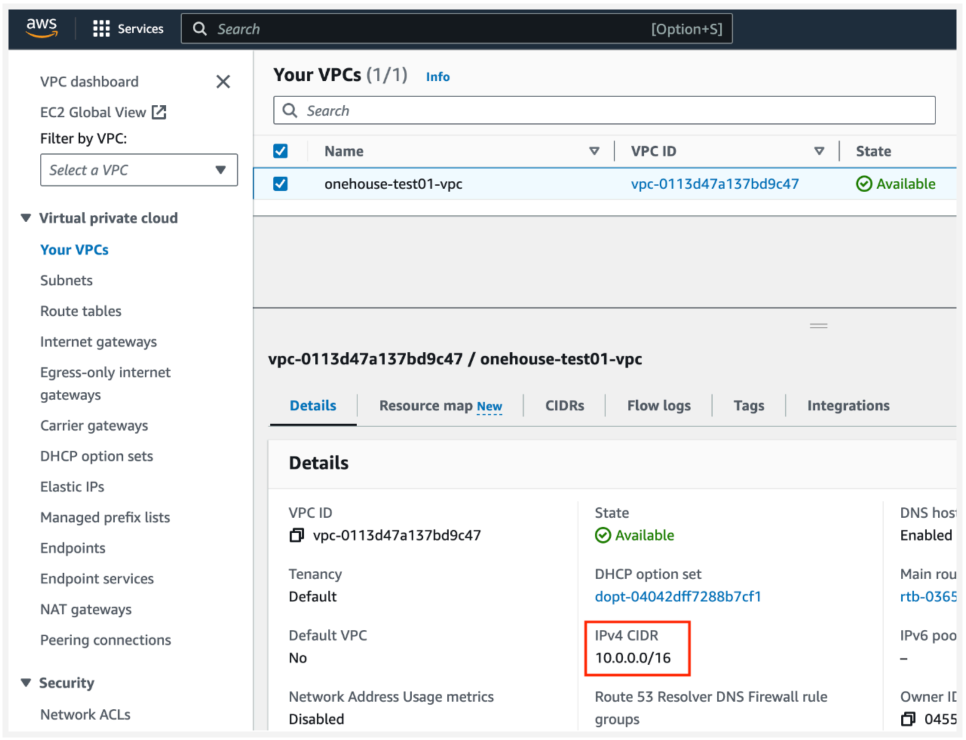
In this section we will create a VPC (virtual private cloud) and a sample Postgres database that is publicly accessible (that is, externally available to desktop database clients so that we can make changes to the data and demonstrate data lake mutations.) If you already have a Postgres database that you can modify, you can skip to the next section.

Note: To minimize costs, the database should be in the same region as your Onehouse installation.

Create a VPC (Virtual Private Cloud) for the database

In this step, we will create a VPC with two availability zones and one public subnet in each zone.

- ☐ Navigate to the AWS VPC dashboard and select 'Your VPCs' in the side menu. Make note of the IPv4 CIDR for this VPC, as highlighted in the figure.



Name	VPC ID	State
onehouse-test01-vpc	vpc-0113d47a137bd9c47	Available

vpc-0113d47a137bd9c47 / onehouse-test01-vpc

Details

VPC ID vpc-0113d47a137bd9c47	State Available	DNS hos Enabled
Tenancy Default	DHCP option set dopt-04042dff7288b7cf1	Main rou rtb-0365
Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 poo –
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups	Owner ID 0455

- ☐ Click the 'Create VPC' button and create a VPC with the following:
  - ☐ Resources to create: 'VPC and more'
  - ☐ Name tag: `acmedb`
  - ☐ IPv4 CIDR: Choose an address range that does not overlap with the CIDR of the onehouse VPC. In this case, the existing VPC has a CIDR of `10.0.0.0/16` so we will use `11.1.0.0/16`.

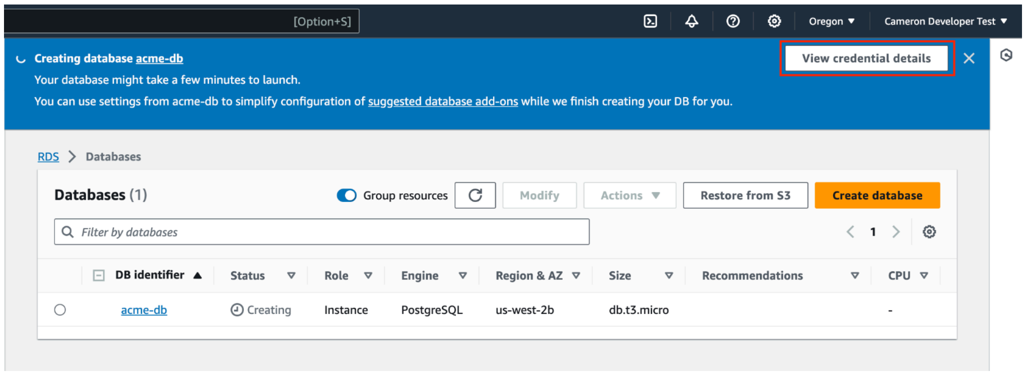
  - ☐ Number of availability zones: 2
  - ☐ Number of public subnets: 2
  - ☐ Number of private subnets: 2
  - ☐ NAT gateways: None
  - ☐ VPC endpoints: None
  - ☐ DNS options:
    - ■ Enable DNS hostnames: Checked
    - ■ Enable DNS resolution: Checked
- ☐ Click into the new VPC and look at the Resource map.
  - ☐ Make note of the availability zones for the subnets and route tables. Also make note of the subnet IDs for the 2 public subnets.

Create the database instance

Before creating the database, we will create a DB subnet group that contains just the two public subnets. Although RDS can automatically create a subnet group during database creation, it will create it with private subnets. We need a subnet group containing only the public subnets to enable the database to be publicly accessible.

- ☐ Create a database subnet group
  - ☐ Navigate to the AWS RDS service.
  - ☐ Click ‘Subnet groups’ in the side menu.
  - ☐ Click ‘Create DB subnet group’ and provide the following details:
    - ■ Name: db-subnet-11-1
    - ■ Description: Public subnets for DB
    - ■ VPC: Choose the VPC you set up for the database.
    - ■ Availability Zones: Select the two availability zones used by the database VPC.
    - ■ Subnets: Select the two public subnets used by the database VPC.
- ☐ Create the database
  - ☐ From within the AWS RDS service, click ‘Databases’ in the side menu.
  - ☐ Click ‘Create database’
  - ☐ Choose ‘Easy create’
  - ☐ Under Configuration choose:
    - ■ Engine type: PostgreSQL
    - ■ DB instance size: Free tier
    - ■ DB-instance identifier: acme-db
    - ■ Master username: postgres
    - ■ Either supply or auto-generate a password.
  - ☐ Click ‘Create database’

- [ ] Note your master password and connection details by clicking the 'View credential details' button when it appears.



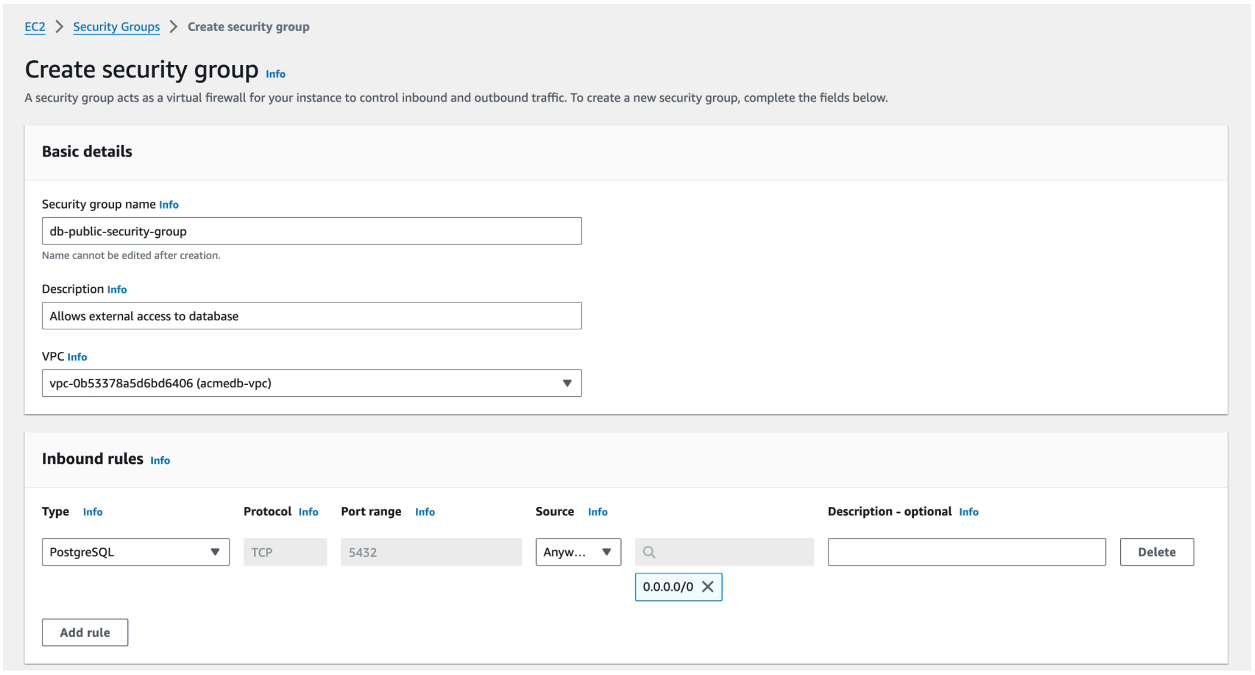
Creating database acme-db
Your database might take a few minutes to launch.
You can use settings from acme-db to simplify configuration of suggested database add-ons while we finish creating your DB for you.

RDS > Databases

Databases (1)

DB identifier	Status	Role	Engine	Region & AZ	Size	Recommendations	CPU
acme-db	Creating	Instance	PostgreSQL	us-west-2b	db.t3.micro		-

- [ ] Create a security group that allows port 5432 to pass in
    - [ ] From the database details page, click the link for the default VPC security group
    - [ ] Click 'Create security group'
    - [ ] Supply the following details:
        - Security group name: db-public-security-group
        - Description: Allows external access to database
        - VPC: Choose the database VPC created earlier
        - Under 'Inbound rules', click 'Add rule'
        - Set the type to: 'PostgreSQL'
        - Source: 'Anywhere-IPv4'
    - [ ] Scroll to the bottom and click: 'Create security group'



EC2 > Security Groups > Create security group

Create security group

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name: db-public-security-group
Name cannot be edited after creation.

Description: Allows external access to database

VPC: vpc-0b53378a5d6bd6406 (acmedb-vpc)

Inbound rules

Type	Protocol	Port range	Source		Description - optional
PostgreSQL	TCP	5432	Anyw...	0.0.0.0/0	

- ☐ Edit the database to make it publicly accessible:
    - ☐ Navigate back to the database page, click 'Modify'
    - ☐ Under 'Connectivity'
        - ■ Remove the default security group, then replace it with the new 'db-public-security-group' security group.
        - ■ Ensure that the subnet group is set to the same that you created earlier.
        - ■ Open the 'Additional configuration' section.
            - ☐ Under 'Public access' click 'Publicly accessible'
    - ☐ Scroll down to the bottom and click 'Continue'
    - ☐ Choose 'Apply immediately' under Schedule modifications
    - ☐ Click 'Modify DB instance'
- ☐ Test the connectivity using your database client of choice. The pgAdmin tool works well; or, if you have the PostgreSQL command line tools installed, you can connect using a command like the following:

```
$ psql -h acme-db.cc64n2qunrtu.us-west-2.rds.amazonaws.com
-U postgres -W -d postgres -p 5432
```

Set up a sample database

In this step, we will create a sample database with simple data that we can update to demonstrate change data capture (CDC) later on.

Download three sample data files and two SQL scripts from:

https://github.com/corourke/acme/tree/main/postgres

Put these files in a convenient location for your database client.

Connect to Postgres as before, then run the following statements as the 'postgres' user:

```
CREATE DATABASE "acme";
GRANT ALL PRIVILEGES on DATABASE acme to postgres;
```

Then, reconnect to the new 'acme' database:

```
$ psql -h acme-db.cc64n2qunrtu.us-west-2.rds.amazonaws.com -U
postgres -W -d acme -p 5432
```

Then create the 'retail' schema:

```
CREATE SCHEMA retail;
SET SEARCH_PATH TO retail;
\i create_tables.sql
```

```
\i load_tables.sql
```

You can confirm that the database is populated by running a few simple queries.

Preparing the AWS Environment for CDC

To be ready to create a Onehouse data source and a stream capture, you need to:

- Ensure that the Amazon RDS parameters are set correctly
- Create a dedicated database user
- Create the heartbeat table
- Create a schema registry in AWS Glue
- Set up a connection between the two VPCs

Ensure that Amazon RDS parameters are set correctly

To verify the status of logical replication, log in to the source PostgreSQL database, then run the following query:

```
SELECT name,setting FROM pg_settings WHERE name IN
('wal_level','rds.logical_replication');
```

You should see the following result:

```
          name                | setting
-------------------------------+---------
 rds.logical_replication       | on
 wal_level                     | logical
(2 rows)
```

If either of these parameters are different, configure an RDS parameter group to set the `'rds.logical_replication'` parameter to 1 per these instructions: [Configure a database parameter group](), then restart the database.

> Note: to avoid filling up your log files and running out of storage space, you may additionally want to set the following parameters in your custom parameter group:
>
> ```
> rds.log_retention_period = 1440
> ```

Create a dedicated database user for Onehouse

Log into the Postgres source data database as the 'postgres' user, then create a dedicated user for Onehouse. Supply your own password:

```
CREATE USER cdc_user WITH ENCRYPTED PASSWORD '<your_password>';
GRANT rds_replication TO cdc_user;
GRANT ALL ON SCHEMA retail TO cdc_user;
```

```
ALTER USER cdc_user SET SEARCH_PATH TO retail;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA retail TO
cdc_user;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA retail TO
cdc_user;
ALTER DEFAULT PRIVILEGES IN SCHEMA retail GRANT ALL PRIVILEGES
ON TABLES TO cdc_user;
ALTER DEFAULT PRIVILEGES IN SCHEMA retail GRANT ALL PRIVILEGES
ON SEQUENCES TO cdc_user;
```

Create a publication that publishes changes in all tables:

```
SET SEARCH_PATH TO retail;
CREATE PUBLICATION alltables FOR ALL TABLES;
```

> Note: If, when executing this statement, you get an error message such as this one:
>
> ```
> WARNING: wal_level is insufficient to publish logical changes
> HINT: Set wal_level to "logical" before creating subscriptions.
> ```
>
> …it means that you need to restart the DB for the custom parameter change to take effect. You can confirm this by checking the wal_level (write-ahead level) in the database; it should be set to 'logical':
>
> ```
> SHOW wal_level;
> ```

Finally, create the heartbeat table:

```
CREATE TABLE onehouse_heartbeat (
    id INTEGER DEFAULT 1 PRIMARY KEY,
    updated_at timestamp DEFAULT CURRENT_TIMESTAMP
    );

INSERT INTO onehouse_heartbeat DEFAULT VALUES ON CONFLICT (id)
DO UPDATE SET updated_at=NOW();
GRANT ALL ON TABLE retail.onehouse_heartbeat TO cdc_user;
```

Create a schema registry in AWS Glue

In this example, we will use the *AWS Glue Schema Registry* to make our datalake data schemas available to other facilities within AWS. These schemas will be populated automatically by Onehouse.

- ☐ In the AWS Management Console, search for and select ‘AWS Glue’.
- ☐ In the left menu, select ‘Stream schema registries’.

- ☐ Click the ‘Add registry’ button.
- ☐ Enter the name, ie ‘Onehouse’, and a description if desired, then click ‘Add registry’.

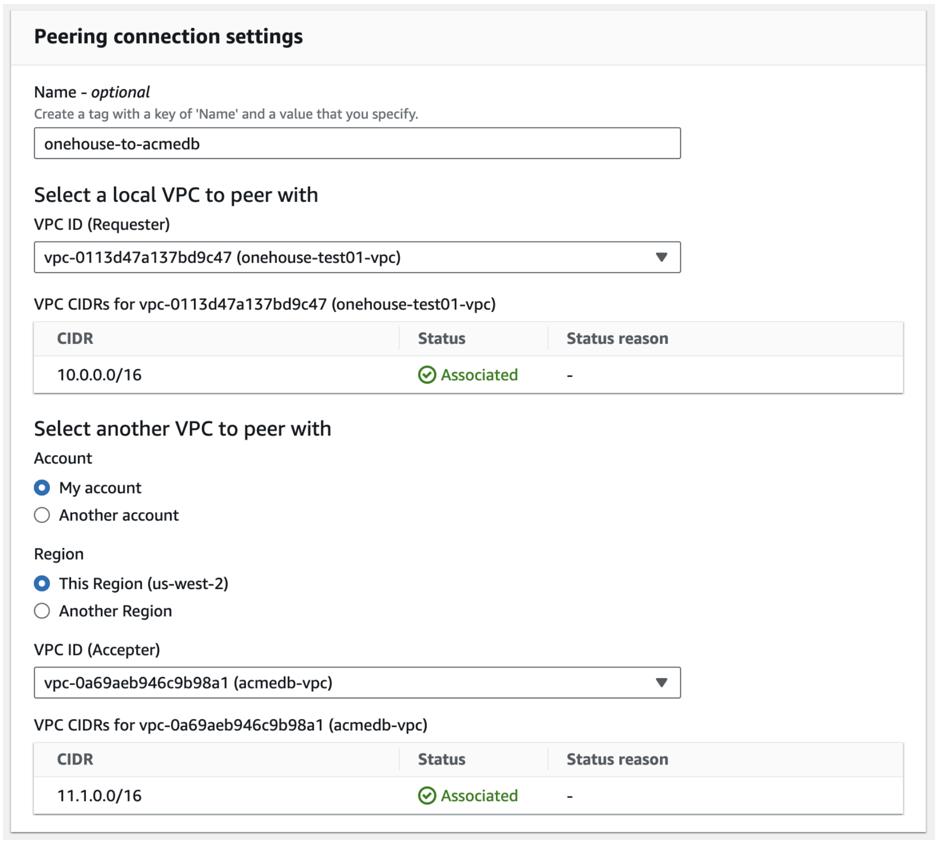
Grant Glue write permissions to the Onehouse IAM role

- ☐ In the AWS console, search for and select IAM.
- ☐ In the menu on the left, select ‘Roles’.
- ☐ Search for roles that start with ‘onehouse-’
- ☐ Find the roles that start with `onehouse-customer-core` and `onehouse-customer-eks-node-role`. For each of these two roles:
    - ☐ Click into the role.
    - ☐ Click the ‘Add permissions’ button and choose ‘Attach policies’
    - ☐ Search for ‘AWSGlue’ and select two policies:
        - ☐ AWSGlueServiceRole
        - ☐ AWSGlueSchemaRegistryFullAccess
    - ☐ Click ‘Add permissions’

Set up a VPC peering connection between the Onehouse VPC and the database VPC

Since the Onehouse infrastructure and our sample Postgres database are in two different VPCs, we need to create a peering connection so that communications can flow between them.

- ☐ In the AWS console, search for and select ‘VPC’.
- ☐ In the menu on the left, select ‘Peering connections’.
- ☐ Click ‘Create peering connection’
    - ☐ Name: onehouse-to-acmedb
    - ☐ Choose the onehouse VPC as the first connection
    - ☐ Choose the database VPC as the second connection
    - ☐ Click ‘Create peering connection’



Peering connection settings

Name - *optional*
Create a tag with a key of 'Name' and a value that you specify.
onehouse-to-acmedb

Select a local VPC to peer with
VPC ID (Requester)
vpc-0113d47a137bd9c47 (onehouse-test01-vpc)

VPC CIDRs for vpc-0113d47a137bd9c47 (onehouse-test01-vpc)

CIDR	Status	Status reason
10.0.0.0/16	✓ Associated	-

Select another VPC to peer with
Account
[✓ My account]
Another account

Region
[✓ This Region (us-west-2)]
Another Region

VPC ID (Accepter)
vpc-0a69aeb946c9b98a1 (acmedb-vpc)

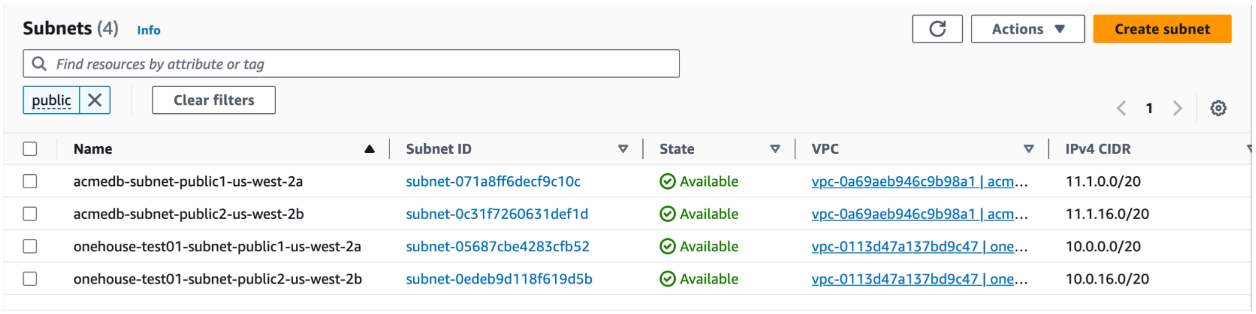
VPC CIDRs for vpc-0a69aeb946c9b98a1 (acmedb-vpc)

CIDR	Status	Status reason
11.1.0.0/16	✓ Associated	-

You must also “accept” the peering connection as an administrator.

Next we need to add routes to the routing tables that connect to the public subnetworks in both VPCs.

- ☐ In the VPC console, select ‘Subnets’ from the menu on the left.
- ☐ Filter subnets by the word ‘public’. You’ll note the public subnets and their respective IPv4 CIDRs.

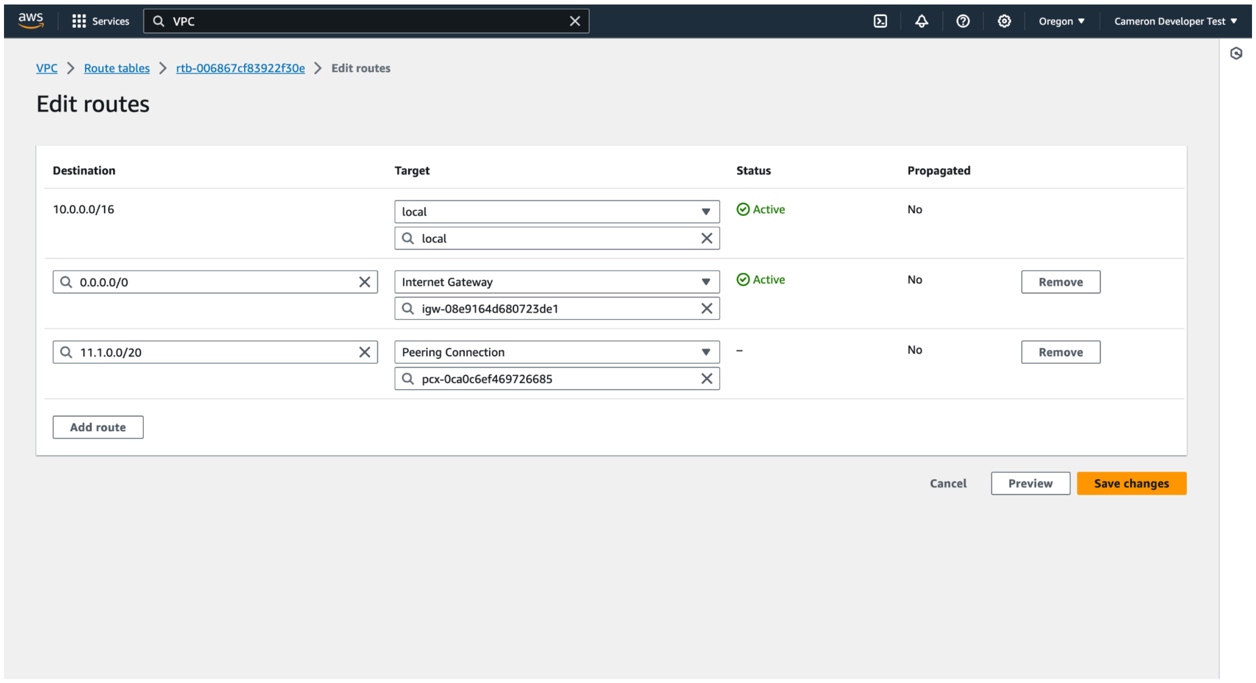


Name	Subnet ID	State	VPC	IPv4 CIDR
acmedb-subnet-public1-us-west-2a	subnet-071a8ff6decf9c10c	✓ Available	vpc-0a69aeb946c9b98a1 | acm…	11.1.0.0/20
acmedb-subnet-public2-us-west-2b	subnet-0c31f7260631def1d	✓ Available	vpc-0a69aeb946c9b98a1 | acm…	11.1.16.0/20
onehouse-test01-subnet-public1-us-west-2a	subnet-05687cbe4283cfb52	✓ Available	vpc-0113d47a137bd9c47 | one…	10.0.0.0/20
onehouse-test01-subnet-public2-us-west-2b	subnet-0edeb9d118f619d5b	✓ Available	vpc-0113d47a137bd9c47 | one…	10.0.16.0/20

  In this case, in the us-west-2a zone, the ‘onehouse’ subnet is using 10.0.0.0/20 and the ‘acmedb’ subnet is using 11.1.0.0/20.

- ☐ On the left menu, select ‘Route tables’ and find route tables for each of the two public subnets. For each one:
  - ☐ Click into the route table, then click ‘Edit routes’

- ☐ Add a route with the destination set to the CIDR of the other VPC. (In this example, for the 10.0.0.0 VPC, we would enter 11.1.0.0/20, and for the 1.1.0.0 VPC, we would enter 10.0.0.0/20.)
- ☐ For the target, choose 'Peering connection' then choose the peering connection we set up earlier.
- ☐ Click 'Save changes'
- ☐ Repeat for the other route table



Edit routes

Destination	Target	Status	Propagated	
10.0.0.0/16	local / local	Active	No	
0.0.0.0/0	Internet Gateway / igw-08e9164d680723de1	Active	No	Remove
11.1.0.0/20	Peering Connection / pcx-0ca0c6ef469726685	–	No	Remove

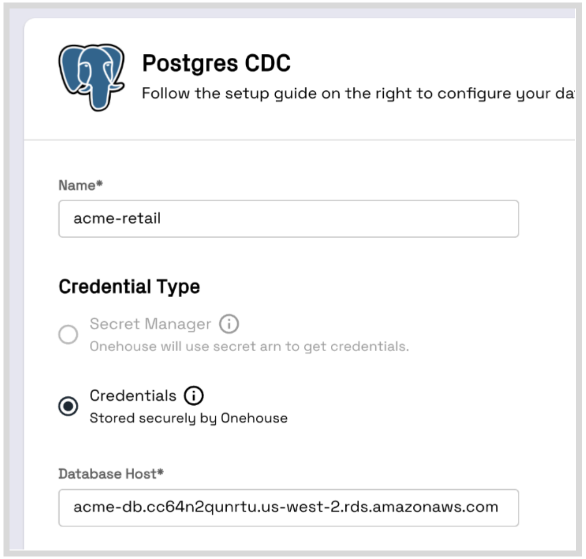
Create a data source and stream capture

With all the setup and configuration out of the way, we can now create a data source (which points to the database) and one or more stream captures (one for each table we want to replicate).

Create a new CDC source in Onehouse

To create a new PostgreSQL CDC source:

- ☐ In the Onehouse console, select 'Sources' on the left menu, then click 'Add New Source'
- ☐ Scroll down the list of sources and choose 'Postgres CDC'
- ☐ Fill in the following fields:
    - ☐ Name: acme-retail-cdc
    - ☐ Credential Type: Stored by Onehouse
    - ☐ Database Host: <the_rds_postgres_hostname>
    - ☐ Database Port: 5432
    - ☐ Database Name: acme
    - ☐ Database User: cdc_user
    - ☐ Database Password: <your_db_user_password>
    - ☐ Schema Registry: AWS Glue Schema Registry: Onehouse



Postgres CDC

Follow the setup guide on the right to configure your da

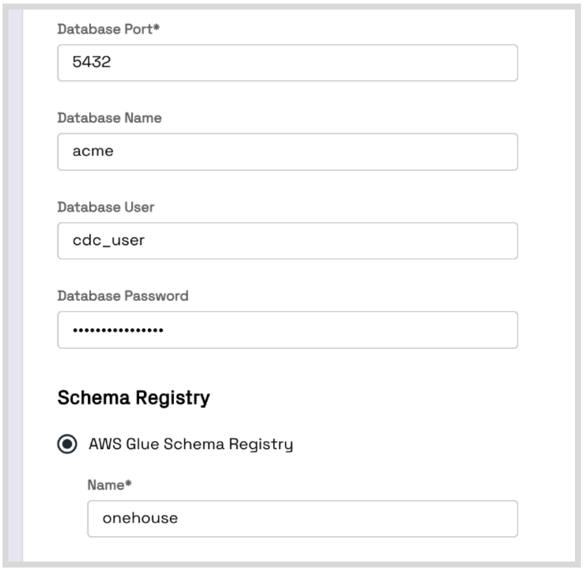
Name*: acme-retail

Credential Type

○ Secret Manager — Onehouse will use secret arn to get credentials.

● Credentials — Stored securely by Onehouse

Database Host*: acme-db.cc64n2qunrtu.us-west-2.rds.amazonaws.com



Database Port*: 5432

Database Name: acme

Database User: cdc_user

Database Password: ••••••••••••••••

Schema Registry

● AWS Glue Schema Registry

Name*: onehouse

Onehouse will perform network connectivity validation tests before adding the new source.

Set up a CDC stream capture in Onehouse

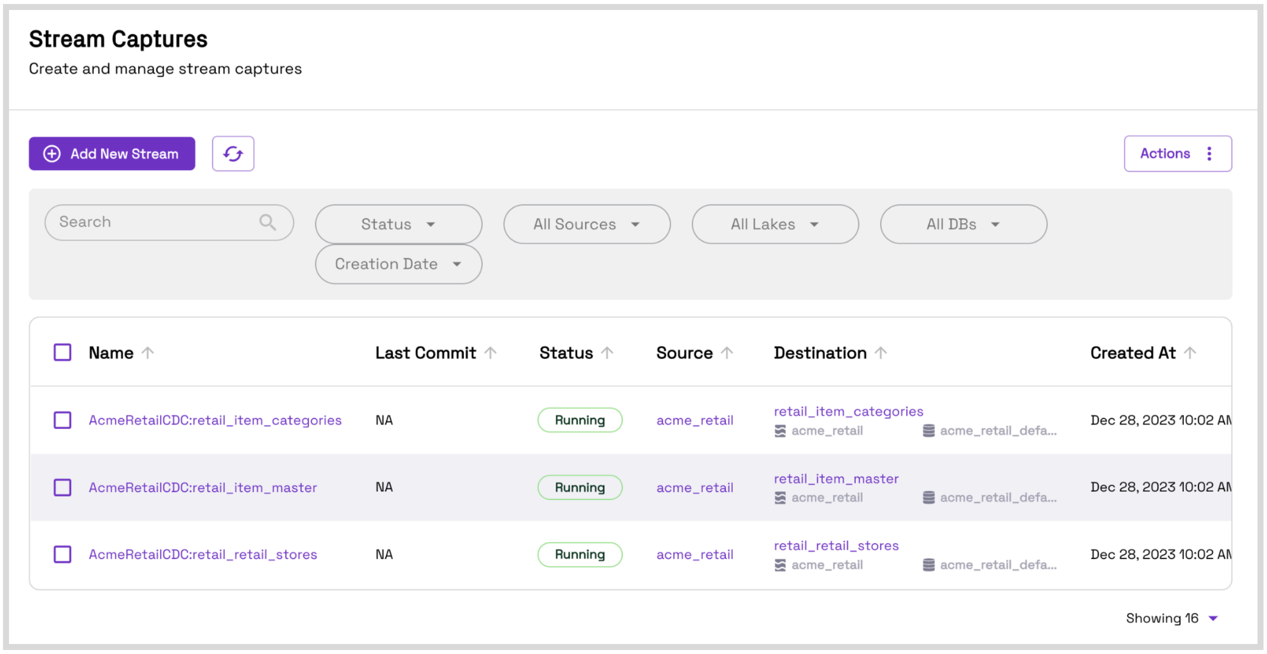
Next, we will create stream captures for each table in our database.

- ☐ In the Onehouse console, select ‘Stream Captures’ on the left menu, then click ‘Add New Stream’ and fill in the form:
- ☐ Step 1: Pick a data source
    - ☐ Name: `AcmeRetailCDC`
    - ☐ Choose the ‘acme-retail’ data source.
- ☐ Step 2: Capture Configs
    - ☐ For ‘Write Mode’ choose Mutable, since we wish to capture updates and deletes.
    - ☐ For Sync Frequency, for this test, choose 5 minutes.
    - ☐ Under Select Tables, you can select all the tables except for ‘retail.scans’ (this will be a high velocity table that we will capture as Append-only, rather than Mutable, in another tutorial.)
    - ☐ For each table, click the ‘Configure’ button.
        - ☐ Under Transformations, the ‘Convert data from CDC format’, CDC Format: Postgres (Debezium) should already be added.
        - ☐ Check that the ‘Record key fields’ has the primary table key added.
        - ☐ ‘Precombine key fields’ should be ‘_event_lsn’.
- ☐ Step 3: Pick data lake and database
    - ☐ Data lake: acme_retail_bronze
    - ☐ Database: acme_retail_default (this was created automatically)
    - ☐ Catalogs: Onehouse, Onetable (both were set up earlier)
- ☐ Click: ‘Start Capturing’

You can now go back to Stream Captures and observe the Status.

It may take approximately 30 minutes the first time you provision a stream capture, due to the time required for Onehouse to set up an Amazon MSK (Managed Streaming for Apache Kafka) cluster and Debezium in the background.

Once the status changes to Running, we can observe the data under the Data tab.



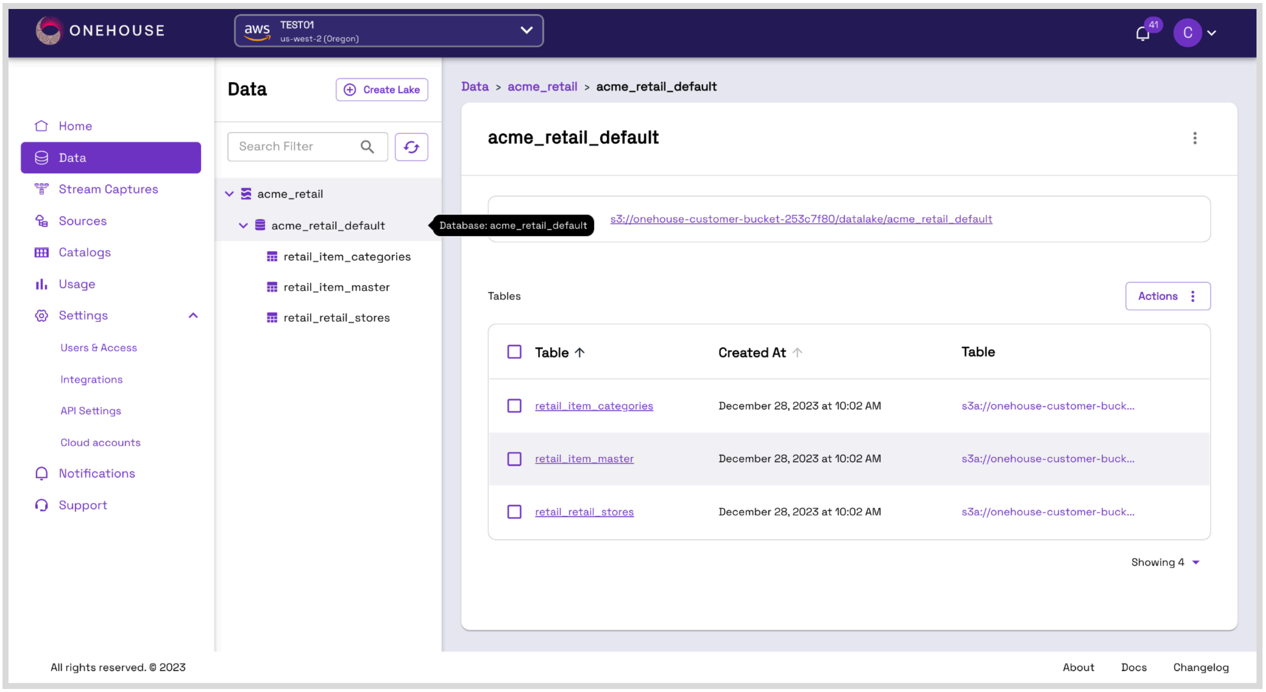
Name	Last Commit	Status	Source	Destination	Created At
AcmeRetailCDC:retail_item_categories	NA	Running	acme_retail	retail_item_categories acme_retail acme_retail_defa...	Dec 28, 2023 10:02 AM
AcmeRetailCDC:retail_item_master	NA	Running	acme_retail	retail_item_master acme_retail acme_retail_defa...	Dec 28, 2023 10:02 AM
AcmeRetailCDC:retail_retail_stores	NA	Running	acme_retail	retail_retail_stores acme_retail acme_retail_defa...	Dec 28, 2023 10:02 AM

Observe Data Ingestion and Mutations

First we will confirm that the seed data has been loaded into the datalake, then we will manually make some database changes and see how the data lake remains synchronized with the database.

Explore the Data tab

This is where we can view the data table statistics, optimizations, and history.



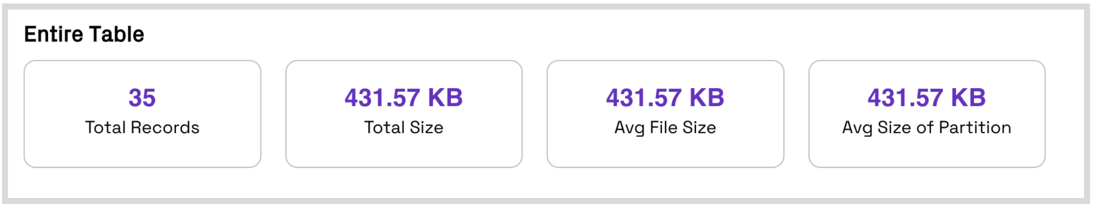
acme_retail_default

s3://onehouse-customer-bucket-253c7f80/datalake/acme_retail_default

Tables

Table ↑	Created At ↑	Table
retail_item_categories	December 28, 2023 at 10:02 AM	s3a://onehouse-customer-buck...
retail_item_master	December 28, 2023 at 10:02 AM	s3a://onehouse-customer-buck...
retail_retail_stores	December 28, 2023 at 10:02 AM	s3a://onehouse-customer-buck...

- ☐ Click the ‘Data’ tab in the menu on the left.
- ☐ Click into the database and click one of the tables.
    - ☐ Note that the table names include the database schema name, prepended to the base table name.
- ☐ On the ‘Overview’ tab:
    - ☐ Observe the total records loaded and data sizes.

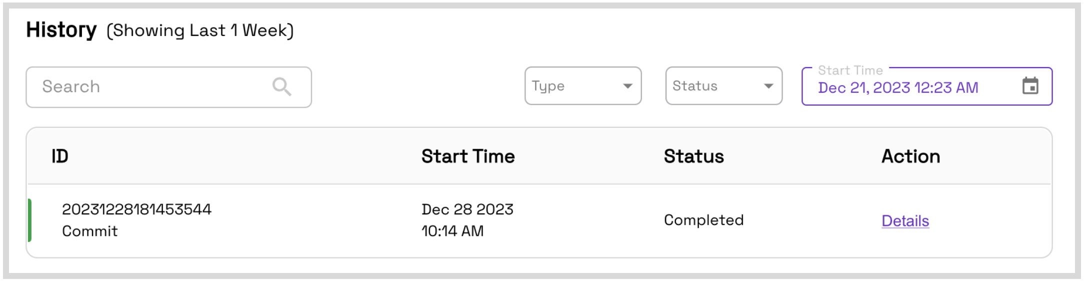


Entire Table

35	431.57 KB	431.57 KB	431.57 KB
Total Records	Total Size	Avg File Size	Avg Size of Partition

- ☐ On the ‘Optimizations’ tab:
    - ☐ We won’t yet see any clustering, compaction, or cleaning activity, but these will come into play as we add more data.

- ☐ On the 'Operations' tab:
    - ☐ Note that this is where we can create and restore point-in-time snapshots.
    - ☐ Also, information about quarantined records can be found here
- ☐ On the 'History' tab:
    - ☐ We can view the data changes made to the data lake over time.



History (Showing Last 1 Week)

Search | Type | Status | Start Time: Dec 21, 2023 12:23 AM

ID	Start Time	Status	Action
20231228181453544 Commit	Dec 28 2023 10:14 AM	Completed	Details

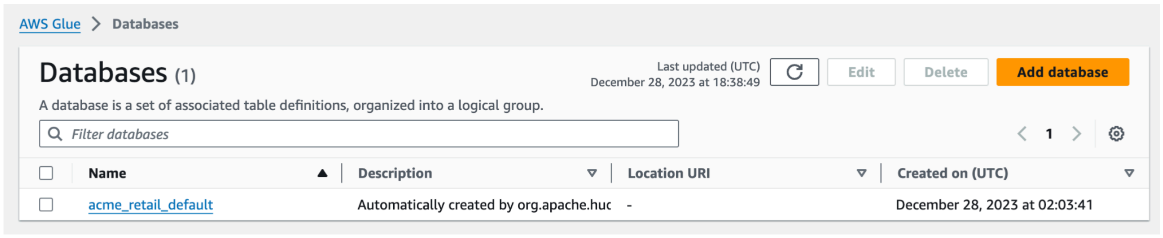
At this point you can confirm that all the seed data has been loaded. Select each of the tables and check the total records loaded:

Item_categories	35
Item_master	32,000
Retail_stores	1,000

Query the datalake

Now we will check the synchronization of metadata in AWS and query the datalake data.

- ☐ In the AWS console, search for 'Glue' and select 'AWS Glue'
- ☐ In the left menu, under 'Data catalog', select 'Databases'. Here you should see a database named 'acme_retail_default'. This mirrors the database shown in the Onehouse UI.



AWS Glue > Databases

Databases (1)

Last updated (UTC) December 28, 2023 at 18:38:49

A database is a set of associated table definitions, organized into a logical group.

Filter databases

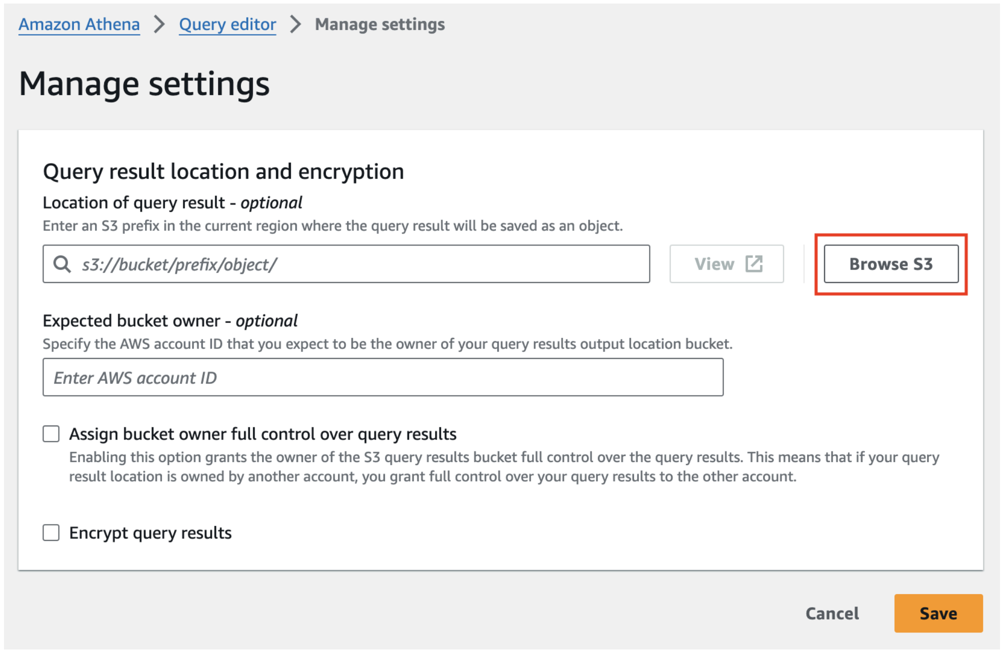
Name	Description	Location URI	Created on (UTC)
acme_retail_default	Automatically created by org.apache.huc	-	December 28, 2023 at 02:03:41

- ☐ Click into the database and you should see the tables you have created. Each table has a version optimized for read-only and a version optimized for read-write operations.

Note: The read-only version is faster, but may not return the freshest data if it has recently been changed. The read-write version always returns the current data.

To query the datalake, we will use Amazon Athena.

- ☐ In the AWS console, search for 'Athena' and select 'Athena'.
- ☐ Open up the left-hand menu, then select 'Query editor'.
- ☐ Before you run your first query, you will be asked to set a query result location in Amazon S3. Click the 'Edit settings' button.
  - ☐ At the 'Manage settings' dialog, click the 'Browse S3' button.



Amazon Athena > Query editor > Manage settings

Manage settings

Query result location and encryption

Location of query result - *optional*

Enter an S3 prefix in the current region where the query result will be saved as an object.

s3://bucket/prefix/object/

Expected bucket owner - *optional*

Specify the AWS account ID that you expect to be the owner of your query results output location bucket.

Enter AWS account ID

☐ Assign bucket owner full control over query results

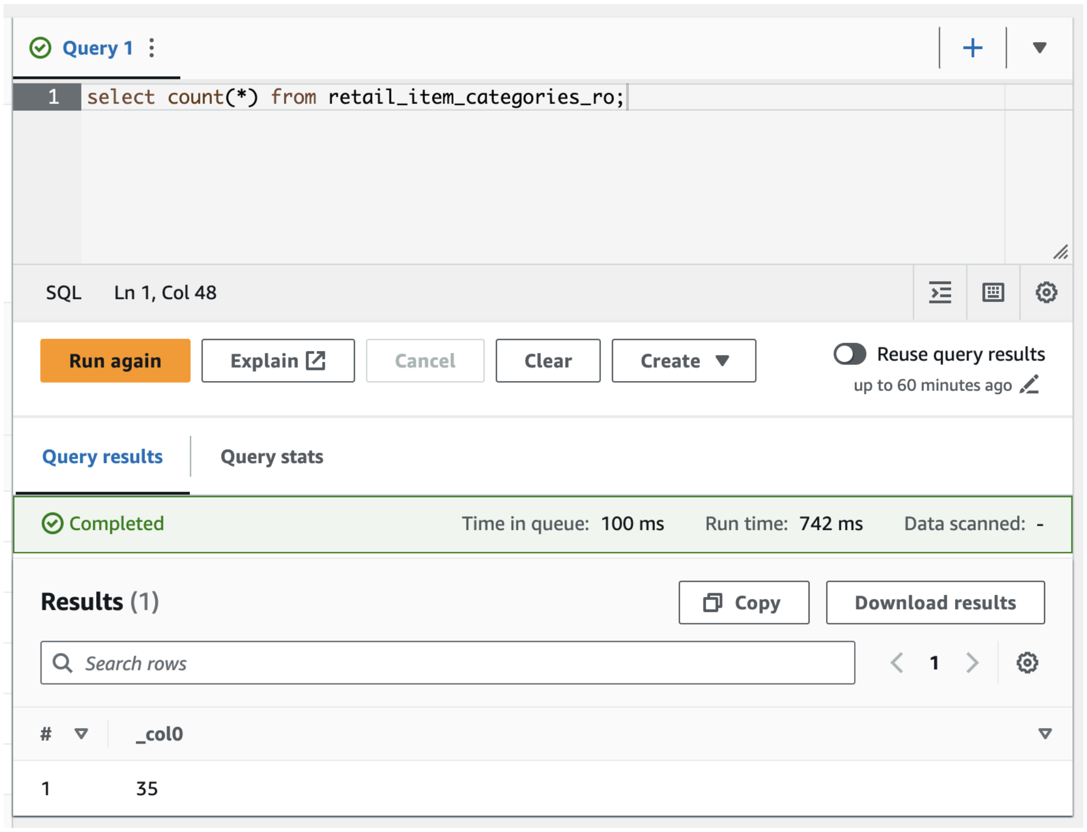
Enabling this option grants the owner of the S3 query results bucket full control over the query results. This means that if your query result location is owned by another account, you grant full control over your query results to the other account.

☐ Encrypt query results

  - ☐ Browse to the 'onehouse-customer-bucket-...', select the 'query_results' folder that we created earlier, then click 'Choose'.
  - ☐ Click 'Save', then click the 'Editor' tab.
- ☐ Now enter a simple query and click 'Run':

```
select count(*) from retail_item_categories_rt;
```

- ☐ Observe the results:



```
select count(*) from retail_item_categories_ro;
```

Completed Time in queue: 100 ms Run time: 742 ms Data scanned: -

#	_col0
1	35

Change the database data

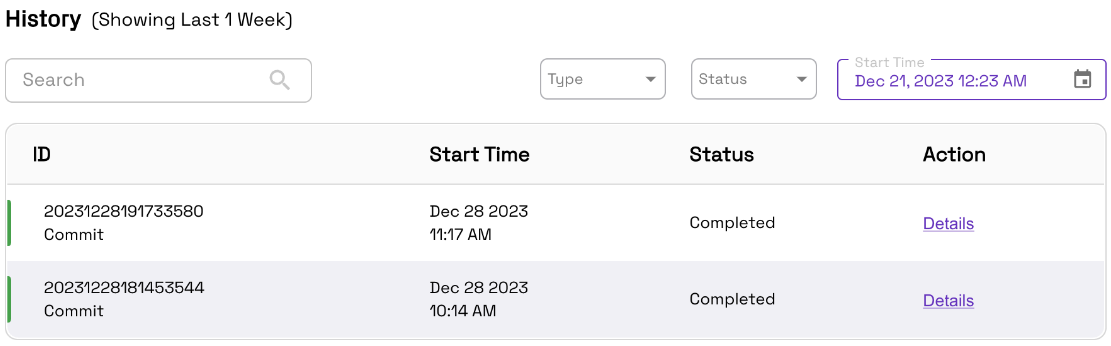
Now we will modify the data in the source PostgreSQL database.

- ☐ Log in to the data using psql or your preferred database client as the 'postgres' user.
- ☐ Issue the following statements:
  - ☐ `INSERT INTO item_categories (category_code, category_name, category_description) VALUES (20900, 'Video Games', 'Video game consoles, games and accessories');`
  - ☐ `UPDATE item_master SET category_code = 20900 where category_code = 20460 and item_id % 8 = 0;`
- ☐ This will create a new product category and re-categorize about 30 of the existing items.

After a few minutes, we can see the changes in Onehouse.

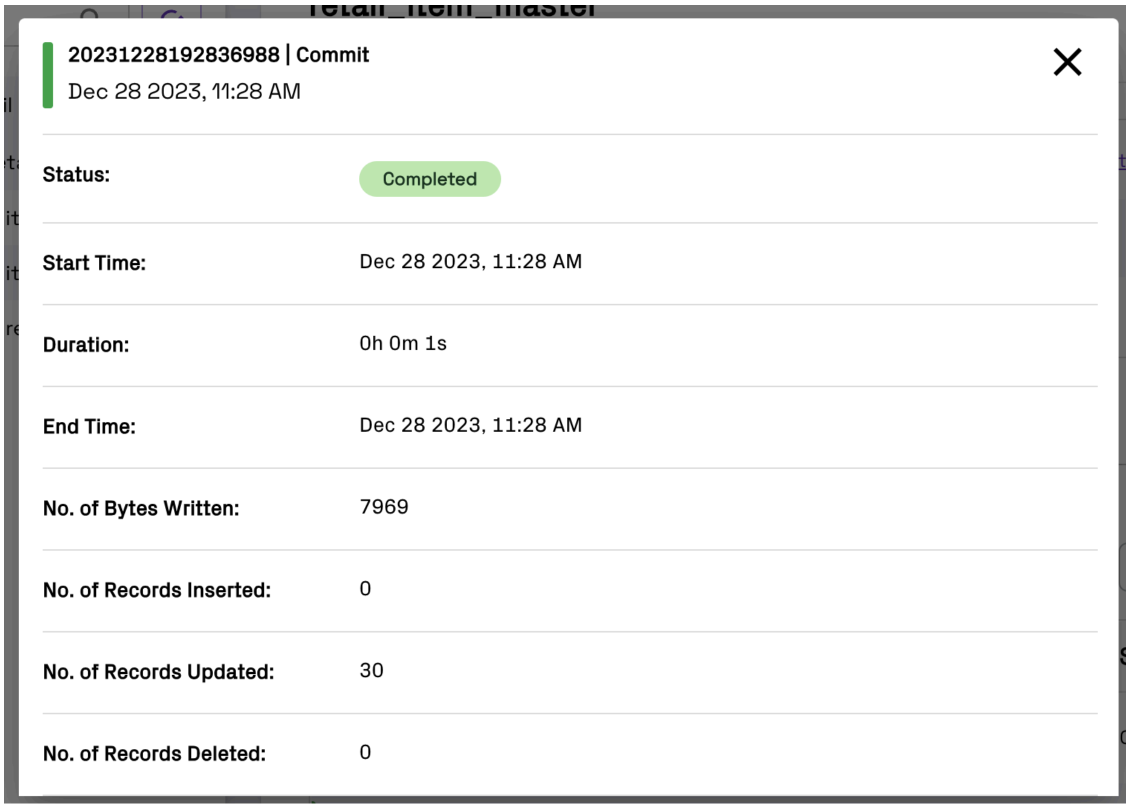
- ☐ In the Data tab, click the 'retail_item_categories' table and the 'Overview' tab; observe that the total number of records is now 36.

- ☐ Click the 'History' tab and see the additional commit:


History (Showing Last 1 Week)

ID	Start Time	Status	Action
20231228191733580 Commit	Dec 28 2023 11:17 AM	Completed	Details
20231228181453544 Commit	Dec 28 2023 10:14 AM	Completed	Details

- ☐ Click the 'retail_item_master' table, go to the History tab, and check for an additional commit. This time, click the 'Details' link to see how many rows were updated.


20231228192836988 | Commit
Dec 28 2023, 11:28 AM

Status:	Completed
Start Time:	Dec 28 2023, 11:28 AM
Duration:	0h 0m 1s
End Time:	Dec 28 2023, 11:28 AM
No. of Bytes Written:	7969
No. of Records Inserted:	0
No. of Records Updated:	30
No. of Records Deleted:	0

- ☐ To confirm that the rows have been updated in the datalake, issue the following query in Athena. The query should return a value of 30.
  `SELECT count(*) FROM retail_item_master_rt WHERE category_code = 20900`

  - ☐ Note: for more information on why we need to specifically use the _rt table to get the freshest data, please see the Hudi concepts doc.

Conclusion

This guide shows how simple it is to use the PostgreSQL change data capture (CDC) feature in Onehouse to set up an end-to-end data pipeline. By leveraging this feature, organizations can replicate data from their PostgreSQL databases to a data lakehouse with near real-time latency, and setup takes just a few clicks! This enables organizations to keep up with the fast-changing demands of the modern business world, where data-driven decision-making is essential for success.

If you want to give Onehouse a try, please visit the Onehouse listing on the AWS Marketplace or contact gtm@onehouse.ai.

Reference

Instantly unlock your CDC PostgreSQL data on a data lakehouse using Onehouse

[Solution] Ingest PostgreSQL CDC Data into the Lakehouse using Onehouse