Date: 5-11-2024

SNOWFLAKE SQL REST API

version: 1.0.0

# **OVERVIEW**

The Snowflake SQL API is a REST API that you can use to access and update data in a Snowflake database.

## **Features:**

You can use this API to develop custom applications and integrations that:

* Perform queries.
* Manage your deployment (e.g. provision users and roles, create tables, etc.).

The Snowflake SQL API provides operations that you can use to:

* Submit SQL statements for execution.
* Check the status of the execution of a statement.
* Cancel the execution of a statement.

**B. Percentage of re-use:**

Approximately 80% of reuse.

**2. Getting Started**

**A. Prerequisites**

## Volt Foundry

* Volt MX Iris
* [Snowflake](https://signup.snowflake.com/?utm_source=google&utm_medium=paidsearch&utm_campaign=ap-in-en-brand-trial-exact&utm_content=go-eta-evg-ss-free-trial&utm_term=c-g-snowflake%20trial-e&_bt=579374819453&_bk=snowflake%20trial&_bm=e&_bn=g&_bg=133380618168&gclsrc=aw.ds&gad_source=1&gclid=Cj0KCQiAoae5BhCNARIsADVLzZfO3xtb5qjXBOmpvJkHApciIFGNik1p4q8miBtgG3-9qs2OUtMXBvQaAjqgEALw_wcB) Account with client id and secret key.

**B. Importing the back-end service**

 To import the Snowflake backend service to Volt Foundry, do the following:

1. Download the **Snowflake back-end service** zip file from [HCL Forge](https://marketplace.hclvoltmx.com/).
2. Sign in to the [HCL Foundry](https://manage.hclvoltmx.com/) Console.
3. On the **Foundry Apps** page, click **IMPORT**.
4. On the **Import App** dialog box, drag and drop the zip file that you downloaded earlier. Alternatively, you can **browse** for the zip file on your system.  
   After the zip file is uploaded, the console displays the default **App Name** and **Version**.
5. Click **IMPORT**.

**C. Obtaining the client id and secret keys.**

1. Start a free trial by creating an account.
2. After creating the account need to provide security integration in the sql Worksheet of snowflake by giving those voltmx callback url and etc.

create or replace security integration DEMOHUB\_INT

type = oauth

enabled = true

oauth\_client = custom

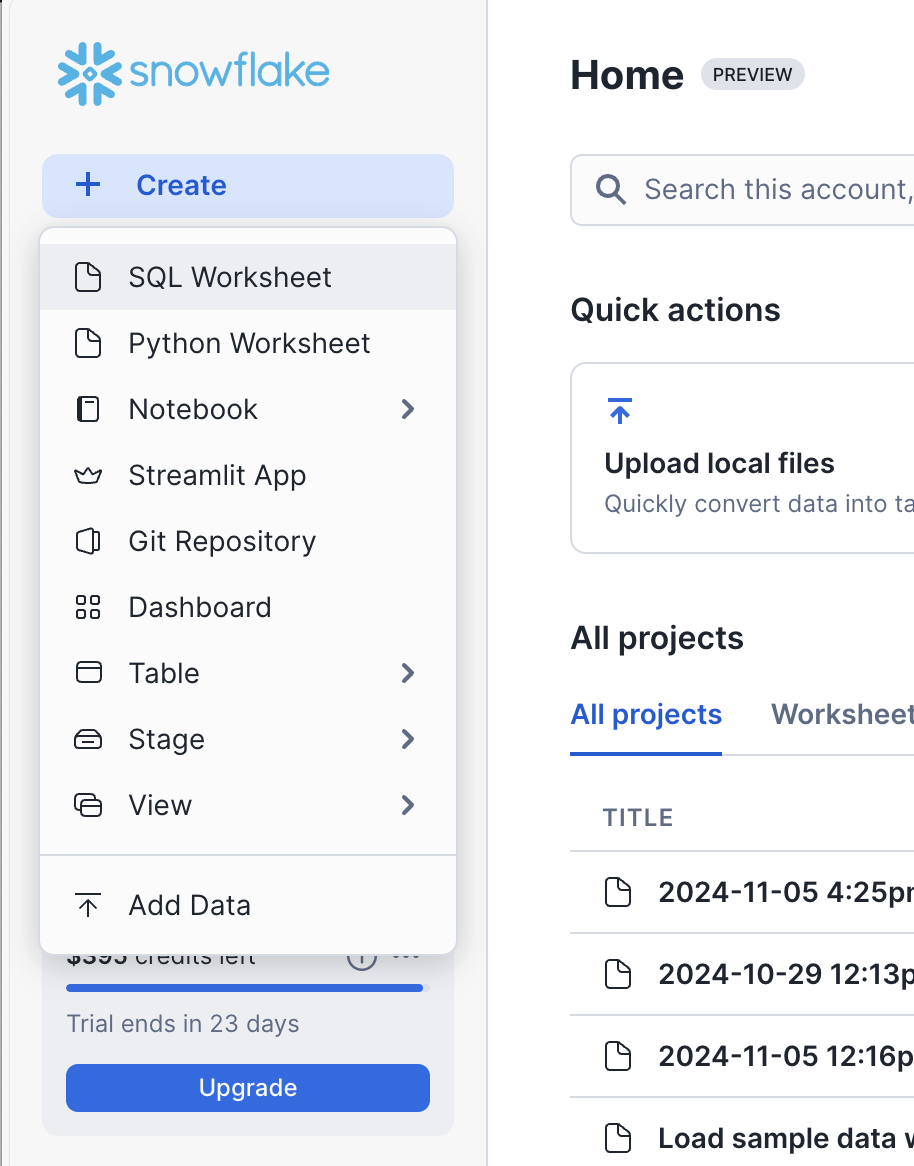
oauth\_client\_type = 'CONFIDENTIAL'

oauth\_redirect\_uri = 'https://100000017.auth.hclvoltmx.net/oauth2/callback'

oauth\_issue\_refresh\_tokens = true

oauth\_refresh\_token\_validity = 86400

;

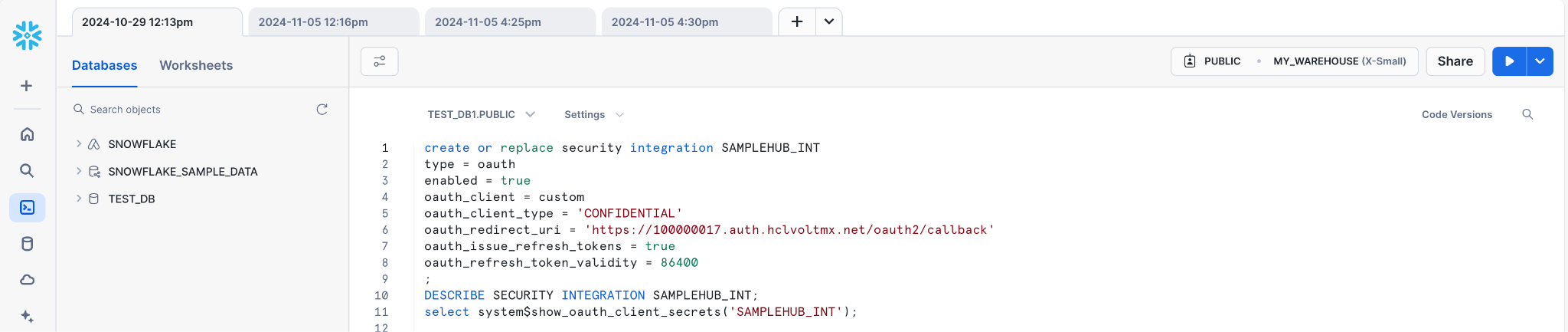


1. After we need to get the keys for that we need to give this query in sql worksheet. We will get client id, authorisation end point, token end point.

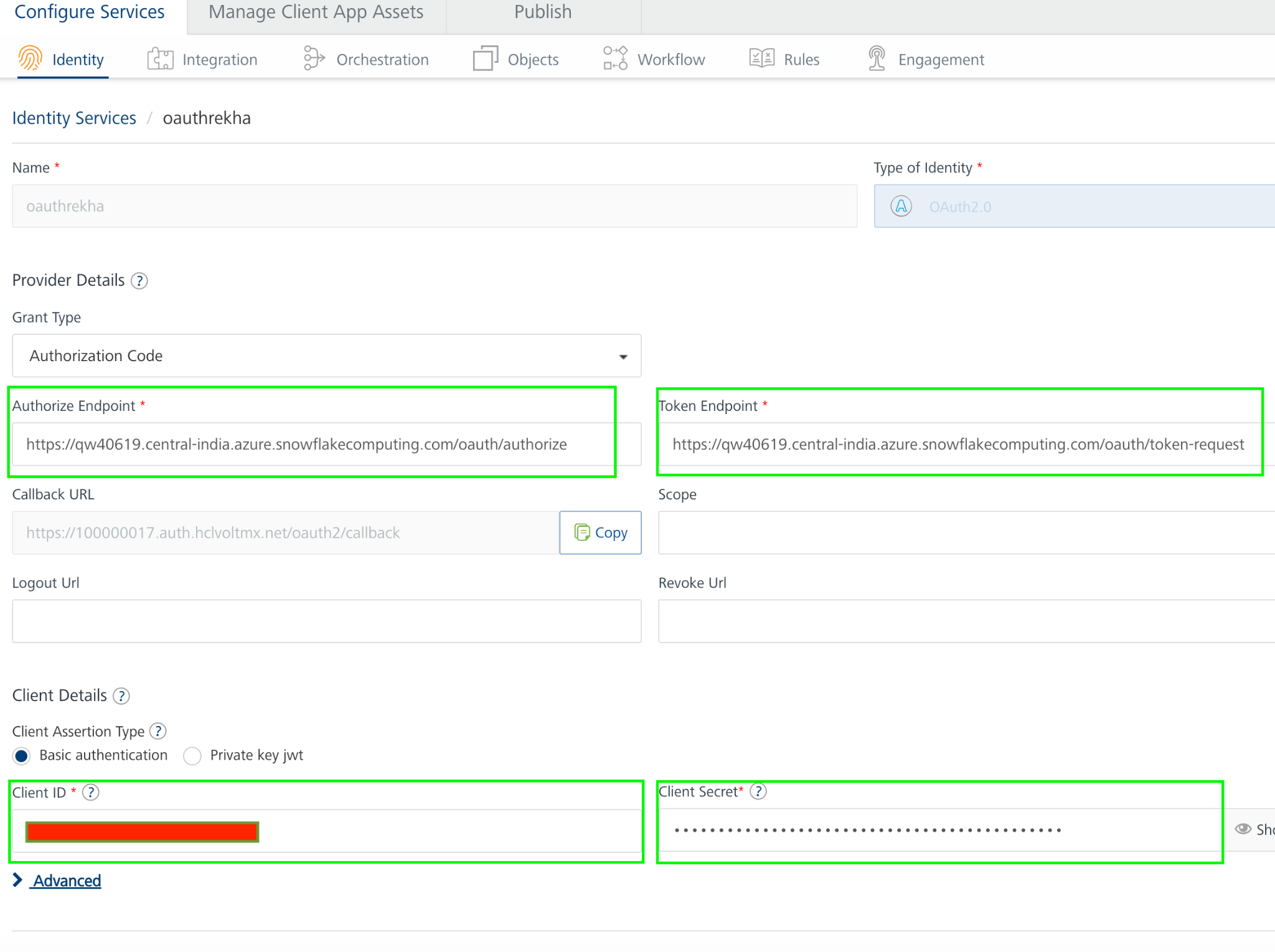
DESCRIBE SECURITY INTEGRATION DEMOHUB\_INT;

4. After that we need to get the secret key for that we need to give this query.

select system$show\_oauth\_client\_secrets('DEMOHUB\_INT');



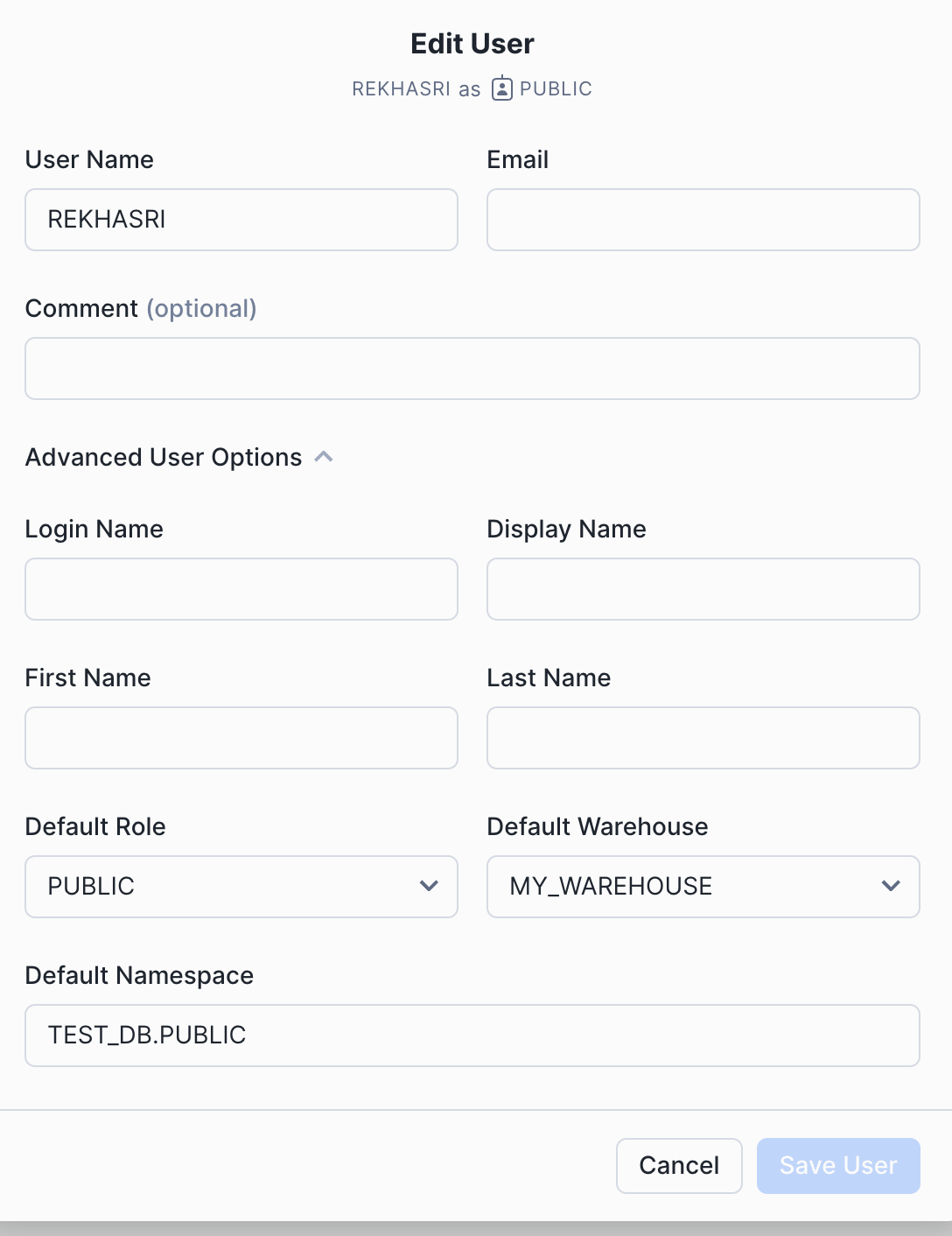
reference screenshot for using the client id, authorization end point, token end point, secret key.



5. By using these we will create Identity service in Foundry.

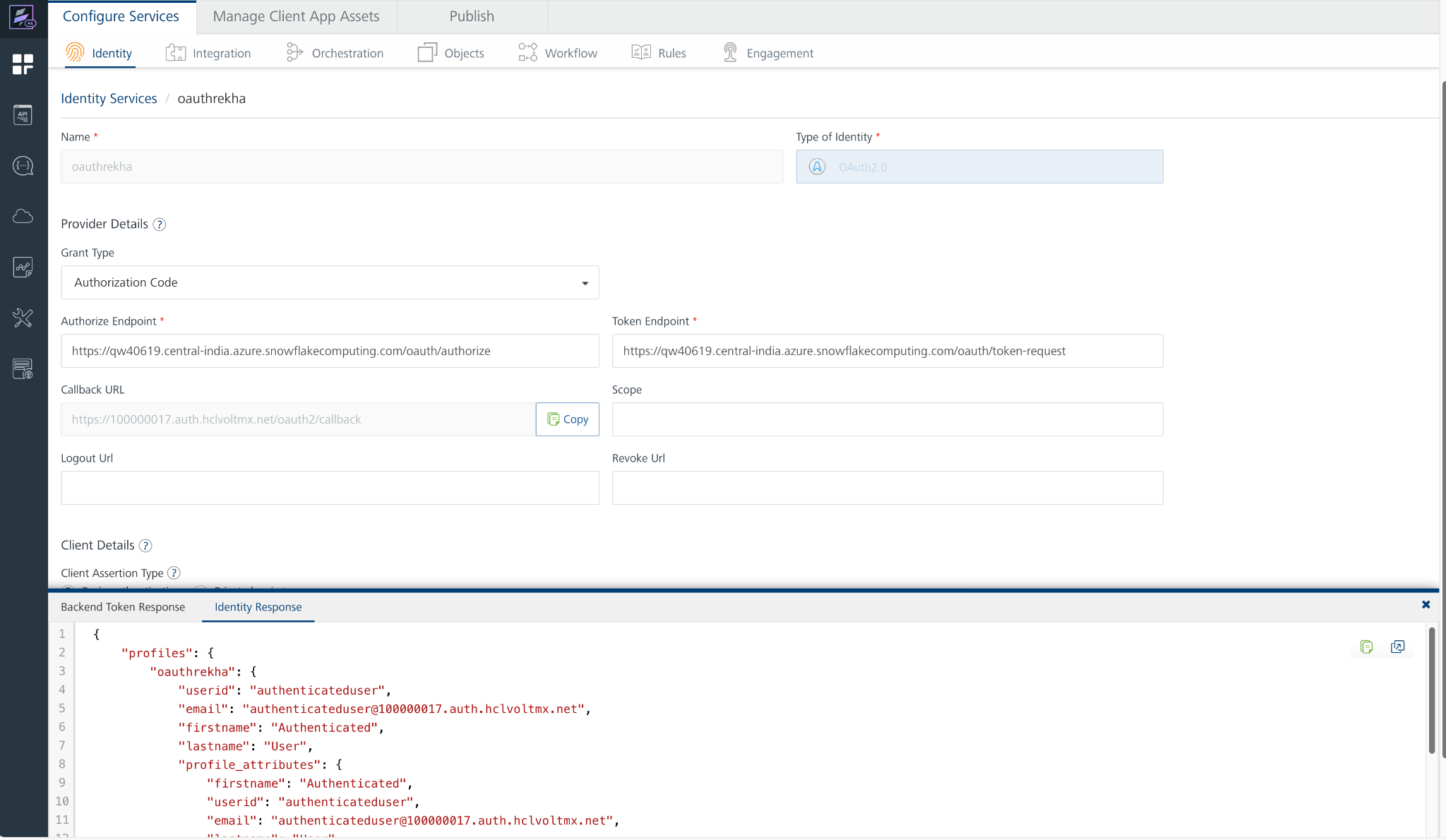
6. To get successful test login of identity service, Default role should be public in the snowflake account; the setup is as shown in below screenshot.





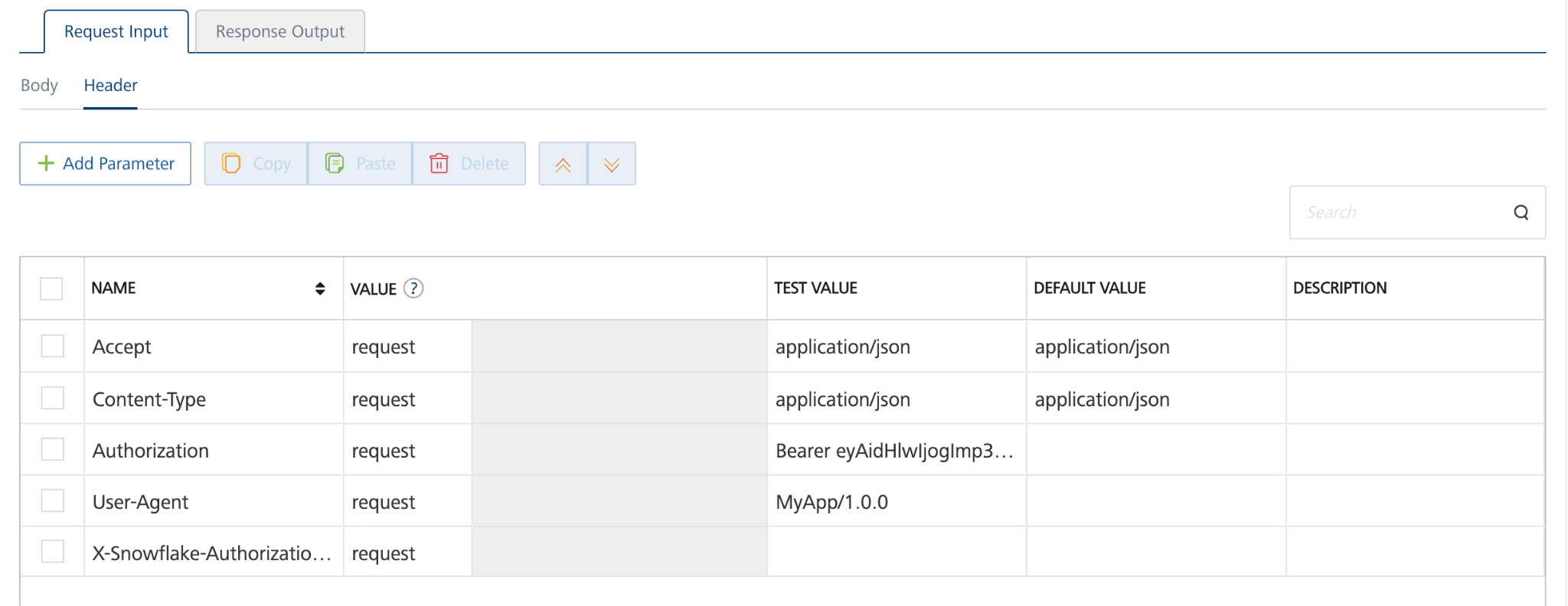
with all respective field details.

6. Reference screenshot of identity service in voltmx foundry



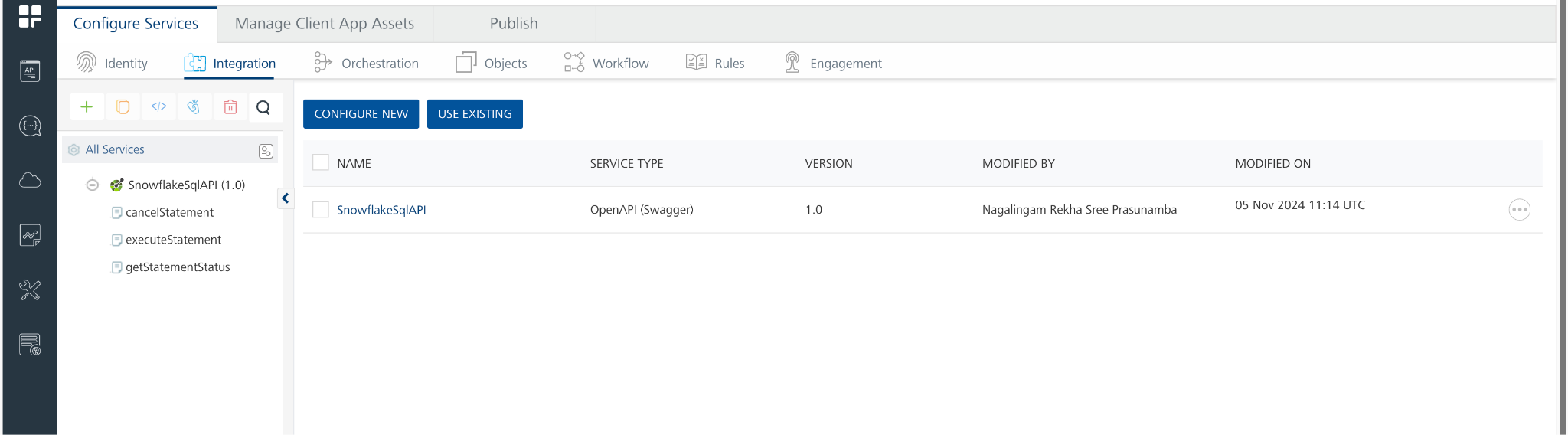
6. And then we will get Bearer token as well after successful test login.

7. In this response we will get claims token, that we can use as Authorization value in the header. Before value we need to add Bearer like below screenshot.

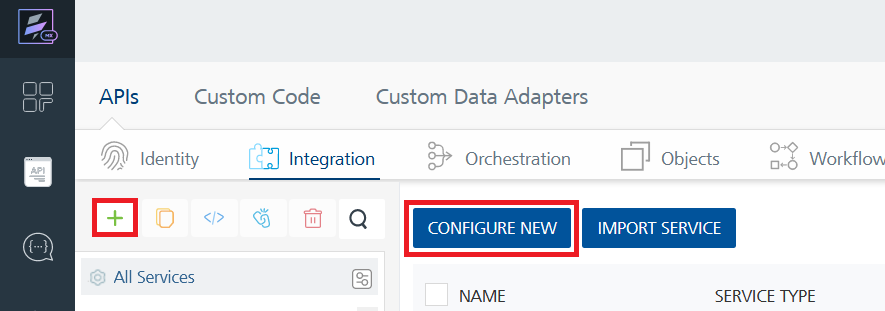


**D. Creating an Integration service**

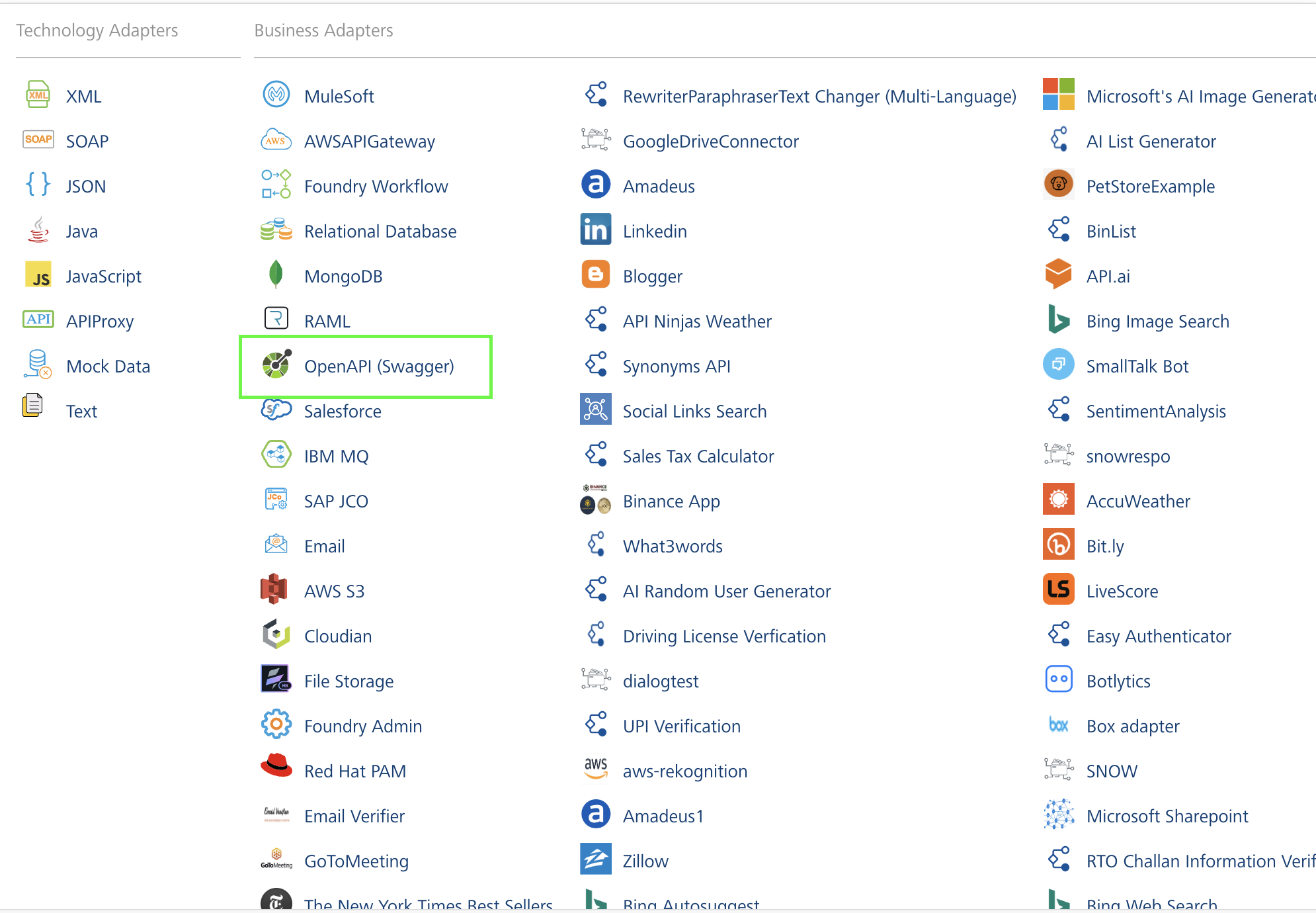
1. Sign in to the [HCL Foundry](https://manage.hclvoltmx.com/) Console.
2. From the **Foundry Apps** page, select the [app that you imported earlier](https://docs.google.com/document/d/1L1h3QDx0yyShIuPwjxp9ra7e8Su24sHk/edit#heading=h.m805qusk7cdw).
3. Under **Configure Services**, on the **Integration** tab, expand the **SnowflakeSqlAPI** service.



4. To create a new service, click the **+** button or the **CONFIGURE NEW** button.

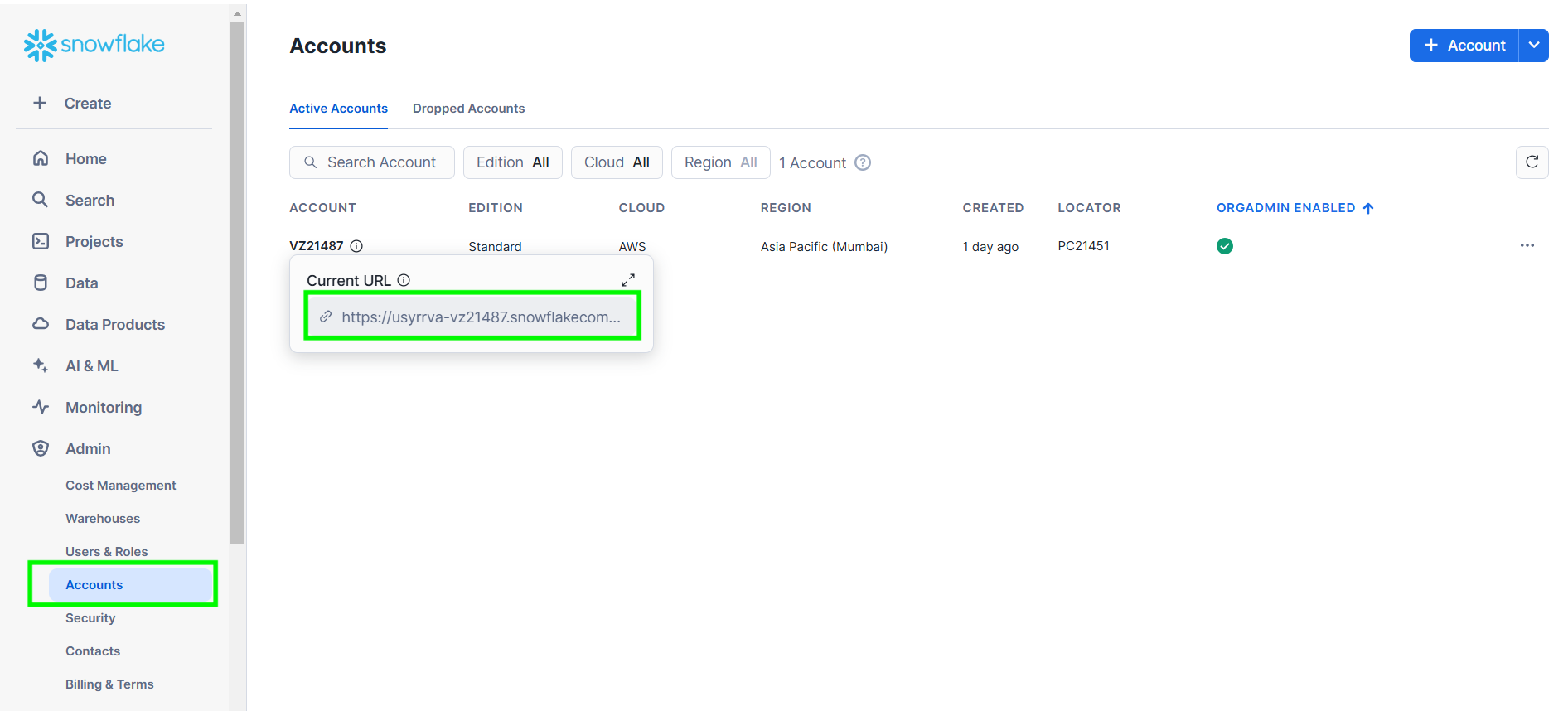


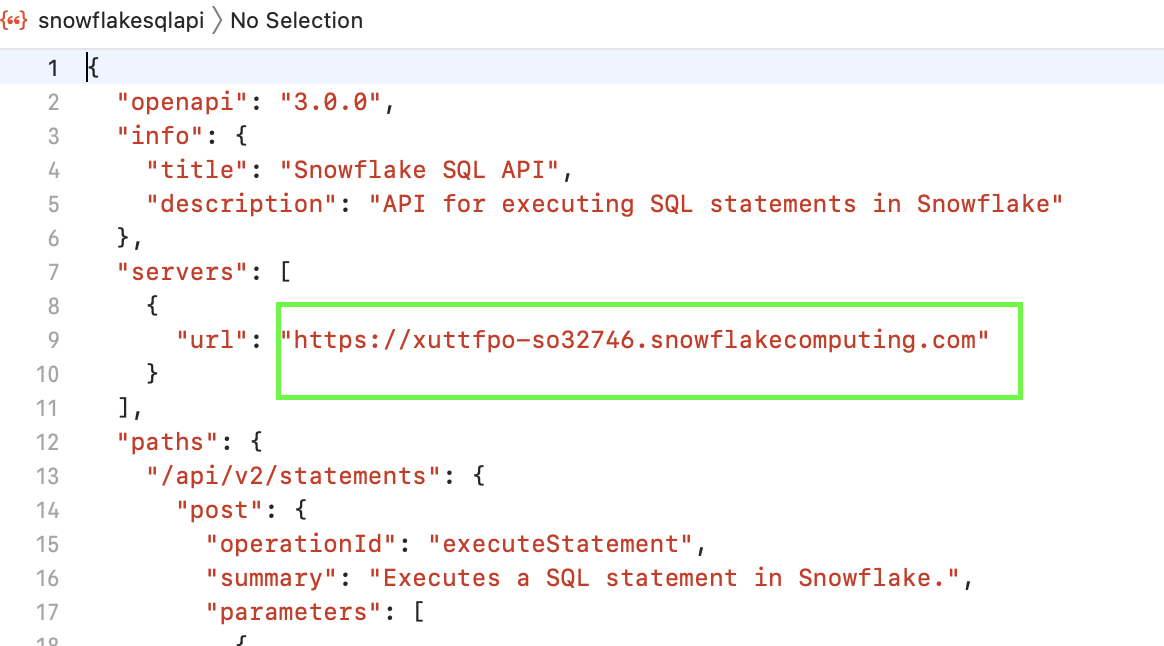
1. On the Service Definition tab, select the service type as **Open** and click **SAVE**.



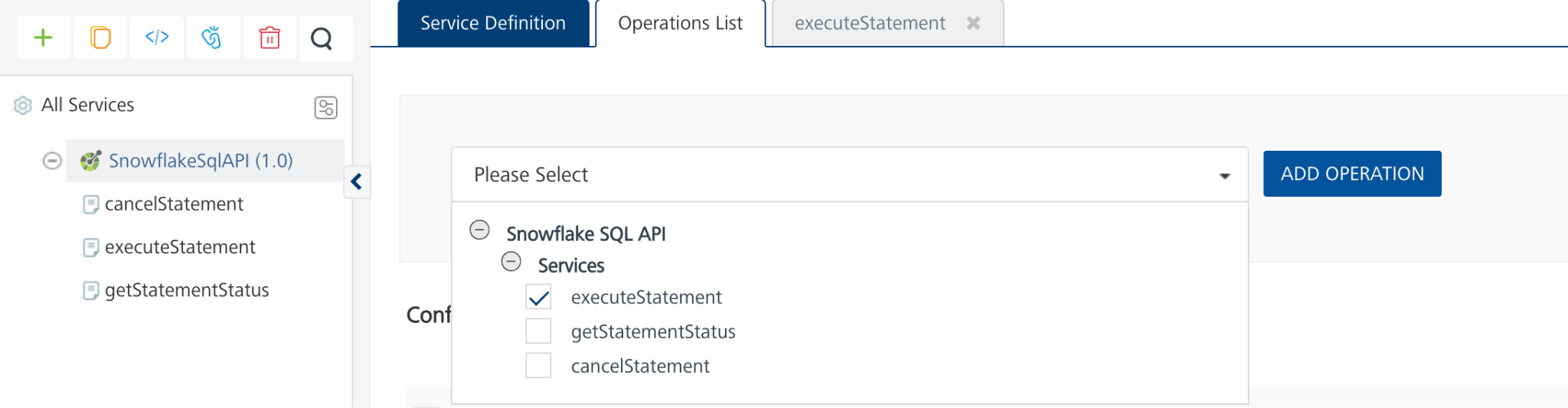
Alternatively, you can also create a Foundry app and create an Integration service inside it.

6. Before adding this to the openapi.json file, replace the host URL with the specific account URL for your respective Snowflake account in json file. This as shown in below screenshot.

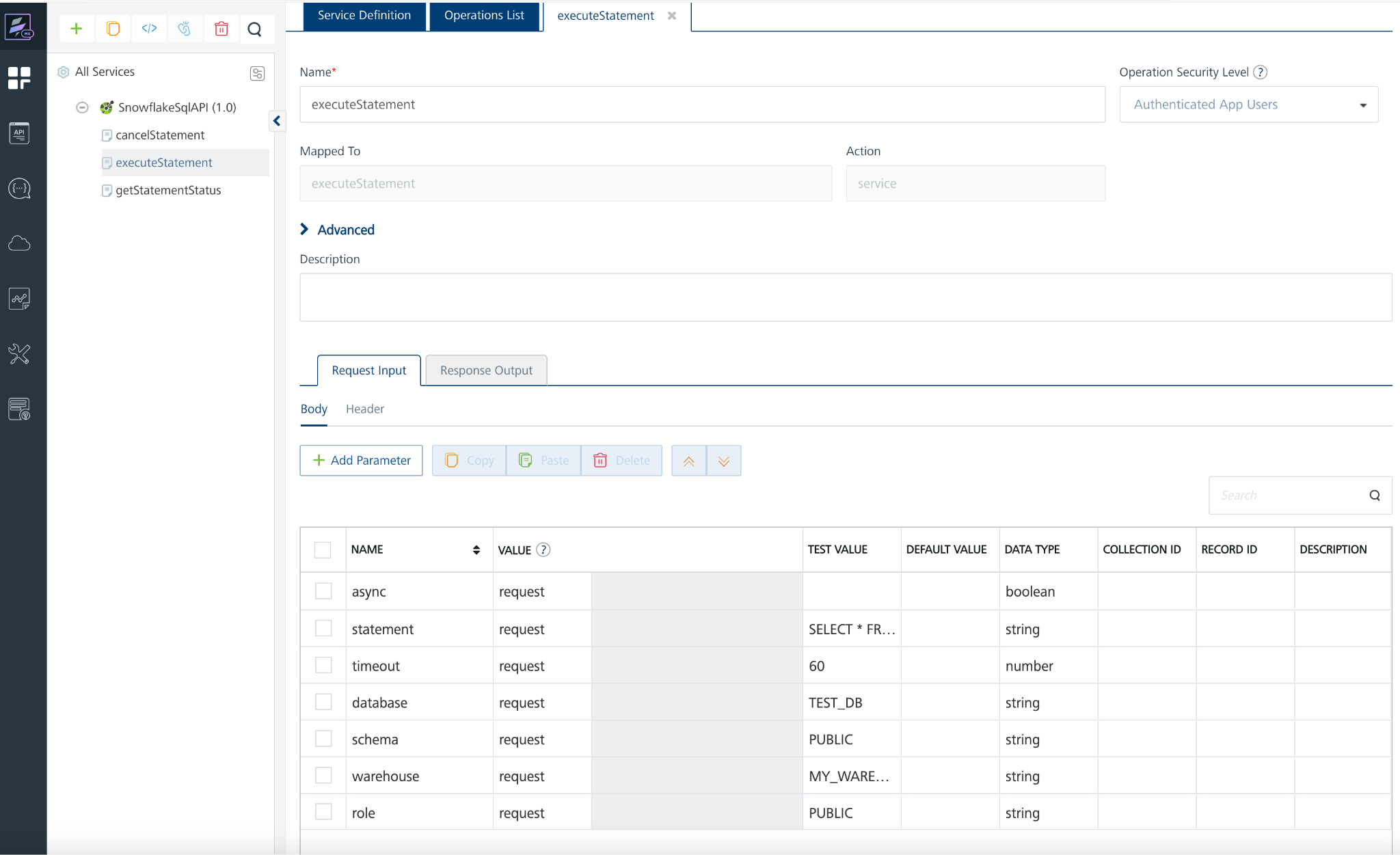




7. From the list of operations, select an operation that you want to test.



8. On the **Request Input** tab, enter a **TEST VALUE** or a **DEFAULT VALUE** for the parameters.

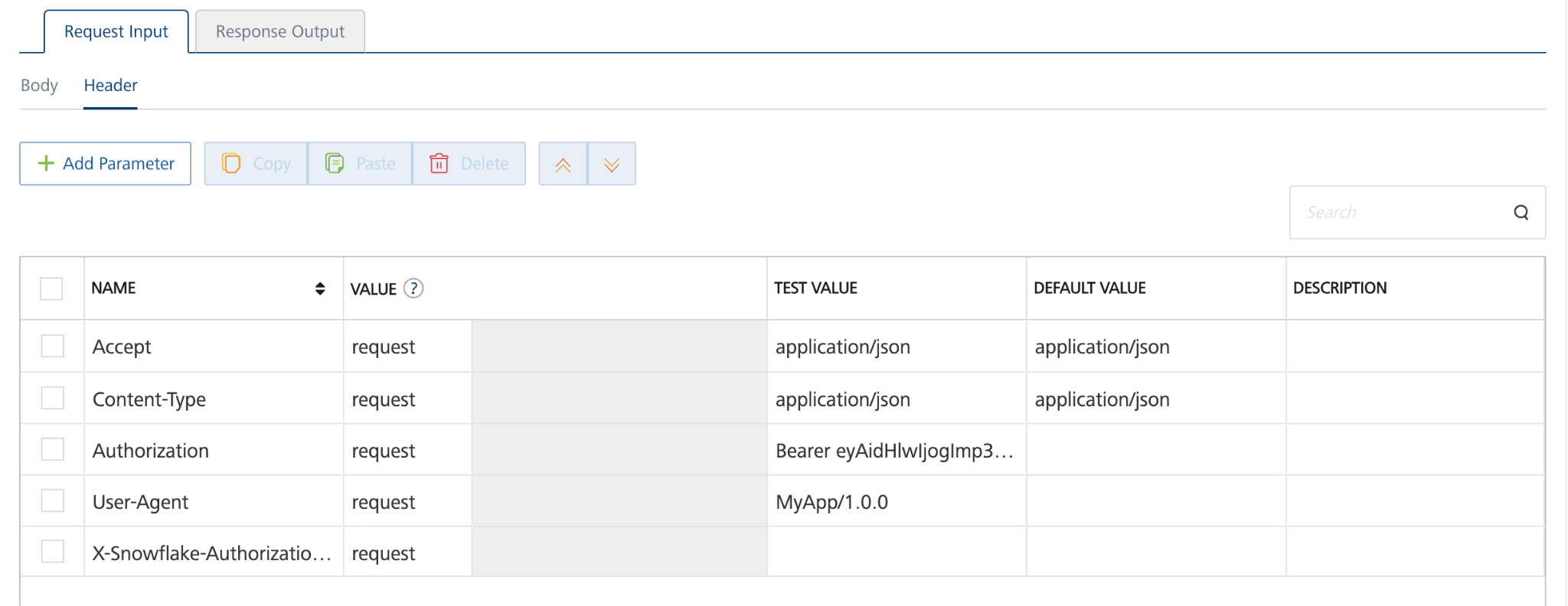


**Note:** To fetch the response onVolt MX Foundry, clear the **Enable pass-through** check box on the **Response Output** tab.

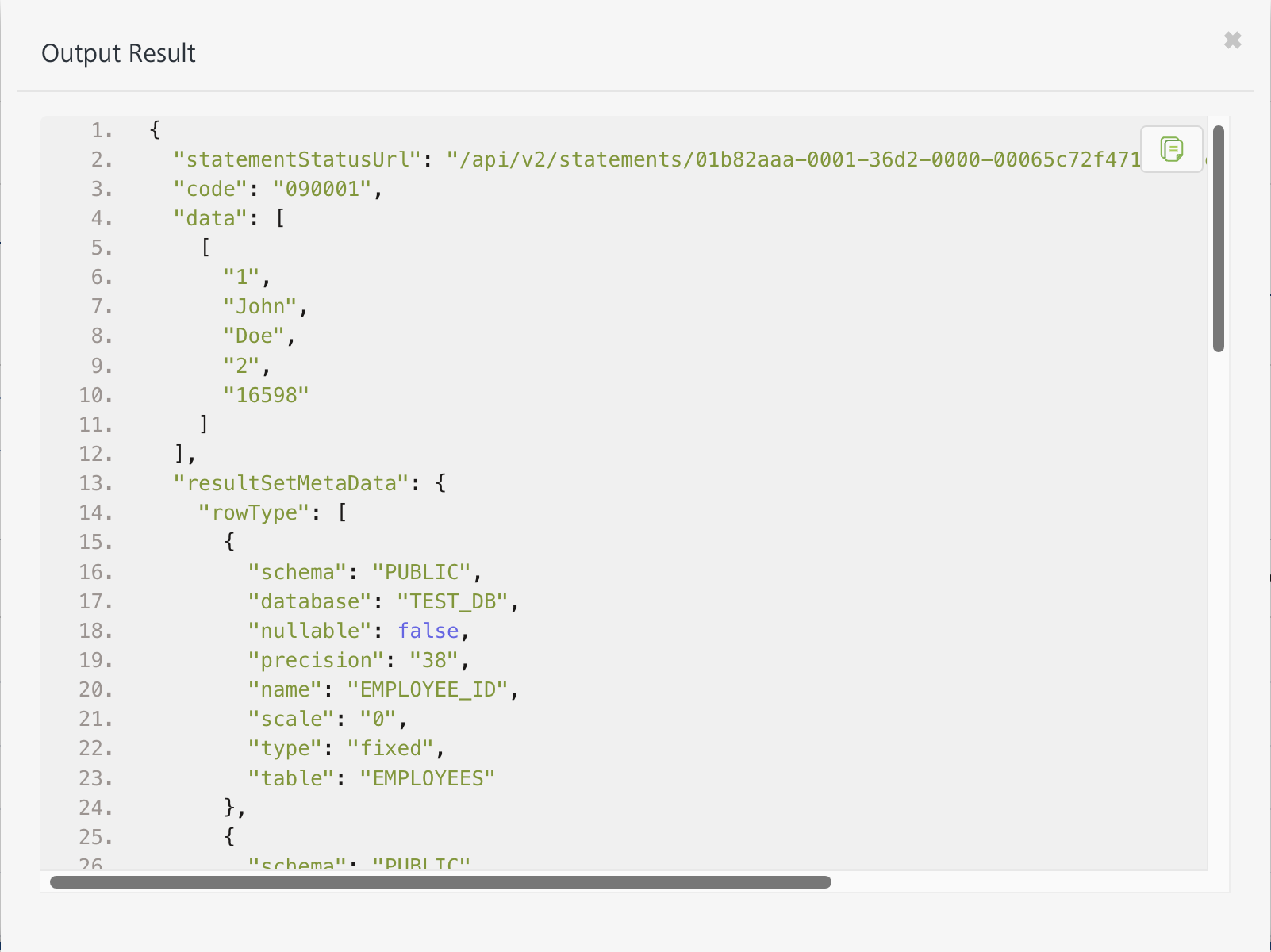
A screenshot of a computer

Description automatically generated

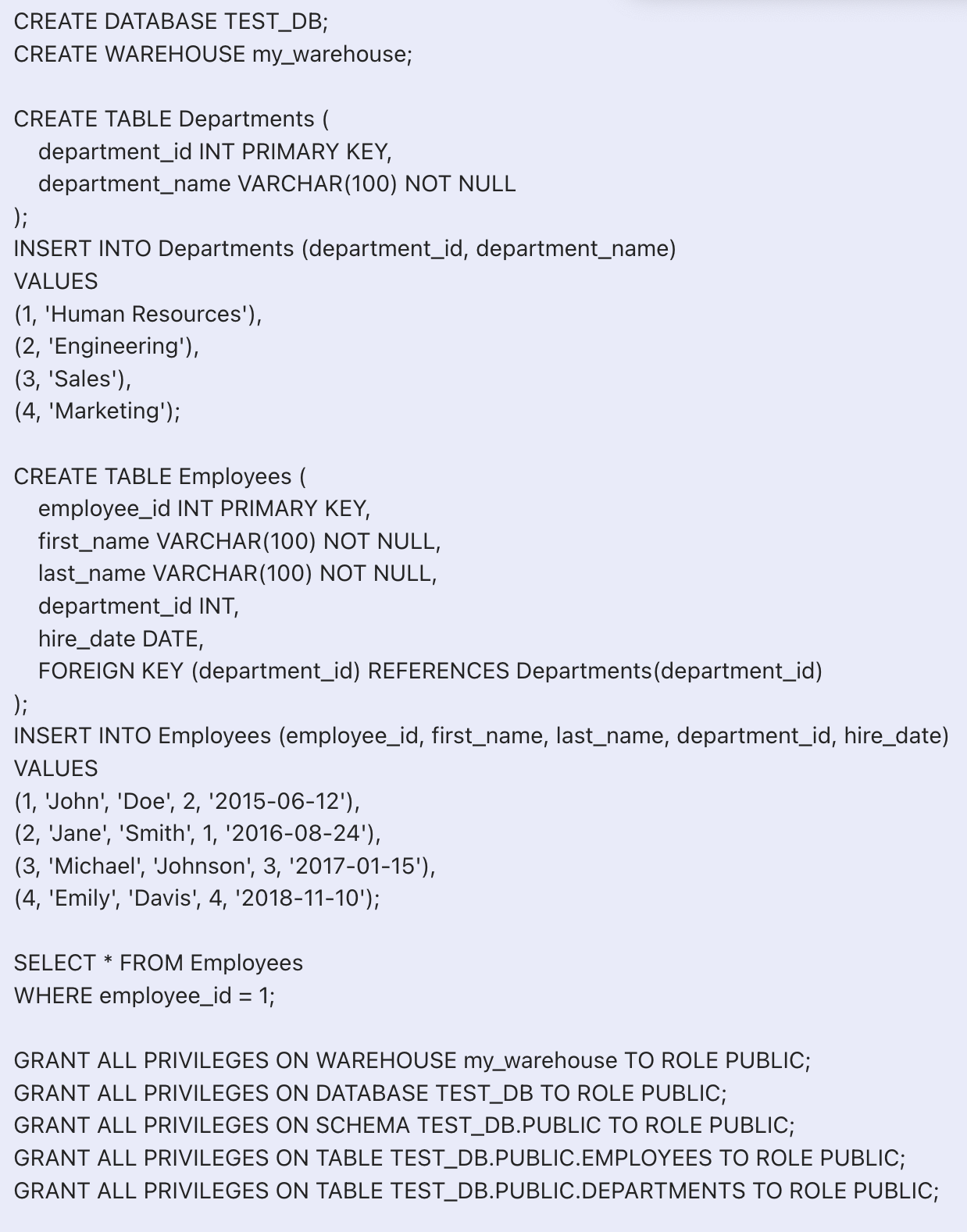
**Request Header:**

**Authorization:** Authorization:The value is the Bearer token that we got from identity service. 

9. Select a run-time environment and click **Save and Fetch Response** to get a response based on your inputs.

**Note:**

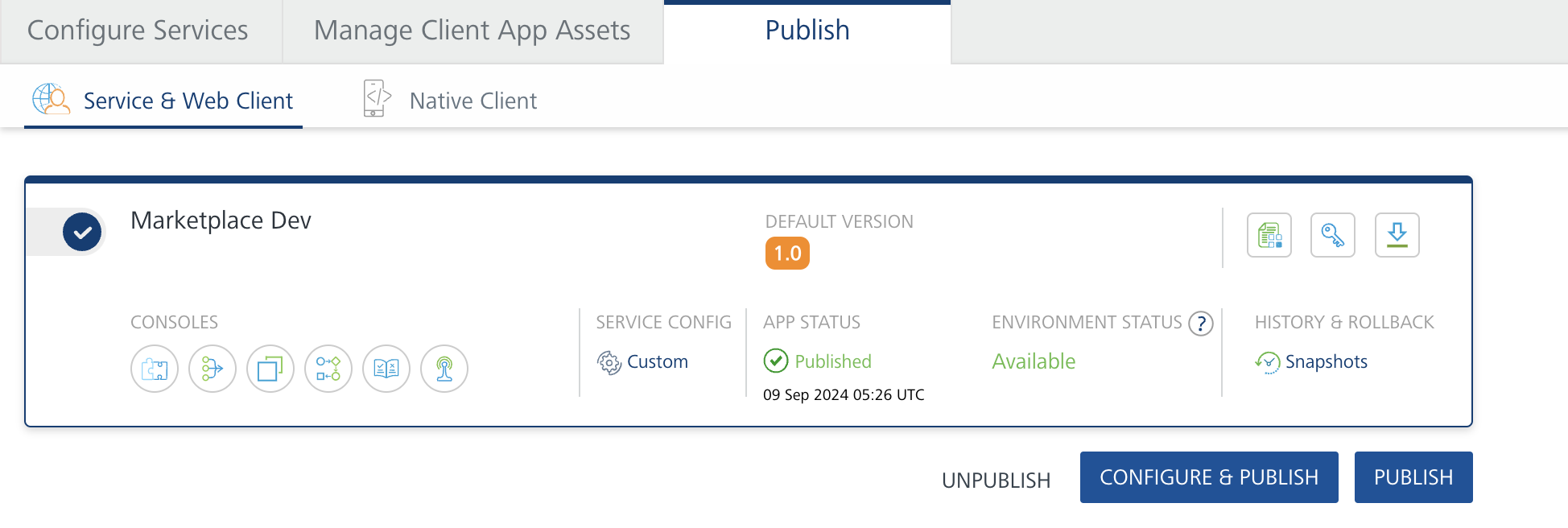
1. To test the operations provided in this link <https://docs.snowflake.com/en/developer-guide/sql-api/index> I have created my own sample database with sample tables in snowflake trial account. Adding reference screenshot for sample database, tables, respective queries to grant privileges.



1. To get successful responses in the foundry, role should be public only in the snowflake account.

## **Publishing your application**

If you want to use the services in client applications, you need to publish an app to a run-time environment. You can create the service (as described above) in an application or import the service into an application and publish the application.



## **References**

**Endpoint Documentation**

<https://docs.snowflake.com/en/developer-guide/sql-api/index>

The SQL API is available at https://*account\_identifier*.snowflakecomputing.com/api, where *account\_identifier* is your [account identifier](https://docs.snowflake.com/en/user-guide/admin-account-identifier).

Beginning with Snowflake version 6.3, the API consists of the /api/v2/statements/ resource and provides the following endpoints:

| **Endpoint** | **Description** |
| --- | --- |
| /api/v2/statements/ | Use this endpoint to [submit SQL statements for execution](https://docs.snowflake.com/en/developer-guide/sql-api/submitting-requests). |
| /api/v2/statements/*statementHandle* | Use this endpoint to [check the status of the execution of a statement](https://docs.snowflake.com/en/developer-guide/sql-api/handling-responses.html#label-sql-api-checking-statement-status). (*statementHandle* is a unique identifier for the statement submitted for execution.) |
| /api/v2/statements/*statementHandle*/cancel | Use this endpoint to [cancel the execution of a statement](https://docs.snowflake.com/en/developer-guide/sql-api/cancelling-requests). |

**1. POST /api/v2/statements operation**:

including the request syntax, query parameters, request headers, request body requirements, and response codes as per your details:

| **Aspect** | **Details** |
| --- | --- |
| **Endpoint** | /api/v2/statements |
| **Method** | POST |
| **Description** | Submits one or more SQL statements for execution. Statements can be executed asynchronously if specified. |
| **Request Syntax** | POST /api/v2/statements |

**Query Parameters**

| **Parameter** | **Type** | **Description** | **Required** |
| --- | --- | --- | --- |
| requestId | UUID | Unique ID of the API request (for resubmitting requests if needed). | Optional |
| async | Boolean | Set to true to execute the statement asynchronously, returning only the statement handle. If false or unspecified, the statement executes synchronously and returns results if completed within 45 seconds; otherwise, only the statement handle is returned if it takes longer. | Optional |
| nullable | Boolean | Set to false to return SQL NULL values as "null" (string) rather than null. By default, SQL NULL values return as null. | Optional |

**Request Headers**

The request must include the standard headers for Snowflake SQL API operations:

| **Header** | **Value** |
| --- | --- |
| Authorization | Bearer <token> |
| Content-Type | application/json |
| Accept | application/json |
| User-Agent | <app\_name/version> |

**Request Body**

The request body must contain a JSON object that specifies the SQL statement(s) to execute. Refer to the [Body of the POST request to /api/v2/statements/](https://docs.snowflake.com/en/developer-guide/sql-api/reference#label-sql-api-reference-post-statements) for more details

.

**Response Codes**

| **Code** | **Description** |
| --- | --- |
| 200 | Statement executed successfully. If a single statement was submitted, the response body contains a ResultSet object with the requested data.  **Note**: If the code field in the response is 391908, the result set is too large to return in full. |

**2. GET /api/v2/statements/{statementHandle}**

endpoint, including the request syntax, path parameters, query parameters, headers, and response codes.

| **Aspect** | **Details** |
| --- | --- |
| **Endpoint** | /api/v2/statements/{statementHandle} |
| **Method** | GET |
| **Description** | Retrieves the status and result data of a previously executed SQL statement using its unique statement handle. |
| **Request Syntax** | GET /api/v2/statements/{statementHandle} |

**Path Parameters**

| **Parameter** | **Type** | **Description** | **Required** |
| --- | --- | --- | --- |
| statementHandle | String | The unique handle of the statement to check. Obtain this from the QueryStatus object returned in response when the statement was first executed. | Required |

**Query Parameters**

| **Parameter** | **Type** | **Description** | **Required** |
| --- | --- | --- | --- |
| requestId | UUID | Unique ID of the API request, useful for resubmitting a request to execute SQL statements. | Optional |
| partition | Integer | Specifies which partition number to return when the result set is partitioned. The size of each partition is determined by Snowflake. | Optional |

**Request Headers**

| **Header** | **Value** | **Description** |
| --- | --- | --- |
| Authorization | Bearer <token> | Access token for authorization. |
| Accept | application/json | Specifies that JSON is expected in response. |
| User-Agent | <app\_name/version> | Identifies the client application. |

**Response Codes**

| **Code** | **Description** |
| --- | --- |
| 200 | The statement was executed successfully. The response body contains a ResultSet object with the requested data if the statement is completed. If the Link header is included, it provides pagination information for accessing additional results. |

**3. POST /api/v2/statements/{statementHandle}/cancel**

endpoint, covering request syntax, path parameters, query parameters, headers, and response codes:

| **Aspect** | **Details** |
| --- | --- |
| **Endpoint** | /api/v2/statements/{statementHandle}/cancel |
| **Method** | POST |
| **Description** | Cancels the execution of an ongoing SQL statement using its unique statement handle. |
| **Request Syntax** | POST /api/v2/statements/{statementHandle}/cancel |

**Path Parameters**

| **Parameter** | **Type** | **Description** | **Required** |
| --- | --- | --- | --- |
| statementHandle | String | Unique handle of the statement to cancel. Obtain this from the QueryStatus object in the response when the statement was executed. | Required |

**Query Parameters**

| **Parameter** | **Type** | **Description** | **Required** |
| --- | --- | --- | --- |
| requestId | UUID | Unique ID for the API request, used for resubmitting requests to execute SQL statements if necessary. | Optional |

**Request Headers**

| **Header** | **Value** | **Description** |
| --- | --- | --- |
| Authorization | Bearer <token> | Access token for authorization. |
| Accept | application/json | Specifies that JSON is expected in response. |
| User-Agent | <app\_name/version> | Identifies the client application. |

**Response Codes**

| **Code** | **Description** |
| --- | --- |
| 200 | Statement execution was successfully cancelled. The response body contains a CancelStatus object, providing information about the cancellation status of the statement. |

**Detailed Explanation:**

To test the statement status and cancellation functionality, you can use a long-running query. Since my sample database lacks queries that exceed 45 seconds, try the following query to artificially induce a delay:

**1. Execute Statement**

sql

WITH Wait AS (

SELECT SYSTEM$WAIT(60) AS Wait\_Result

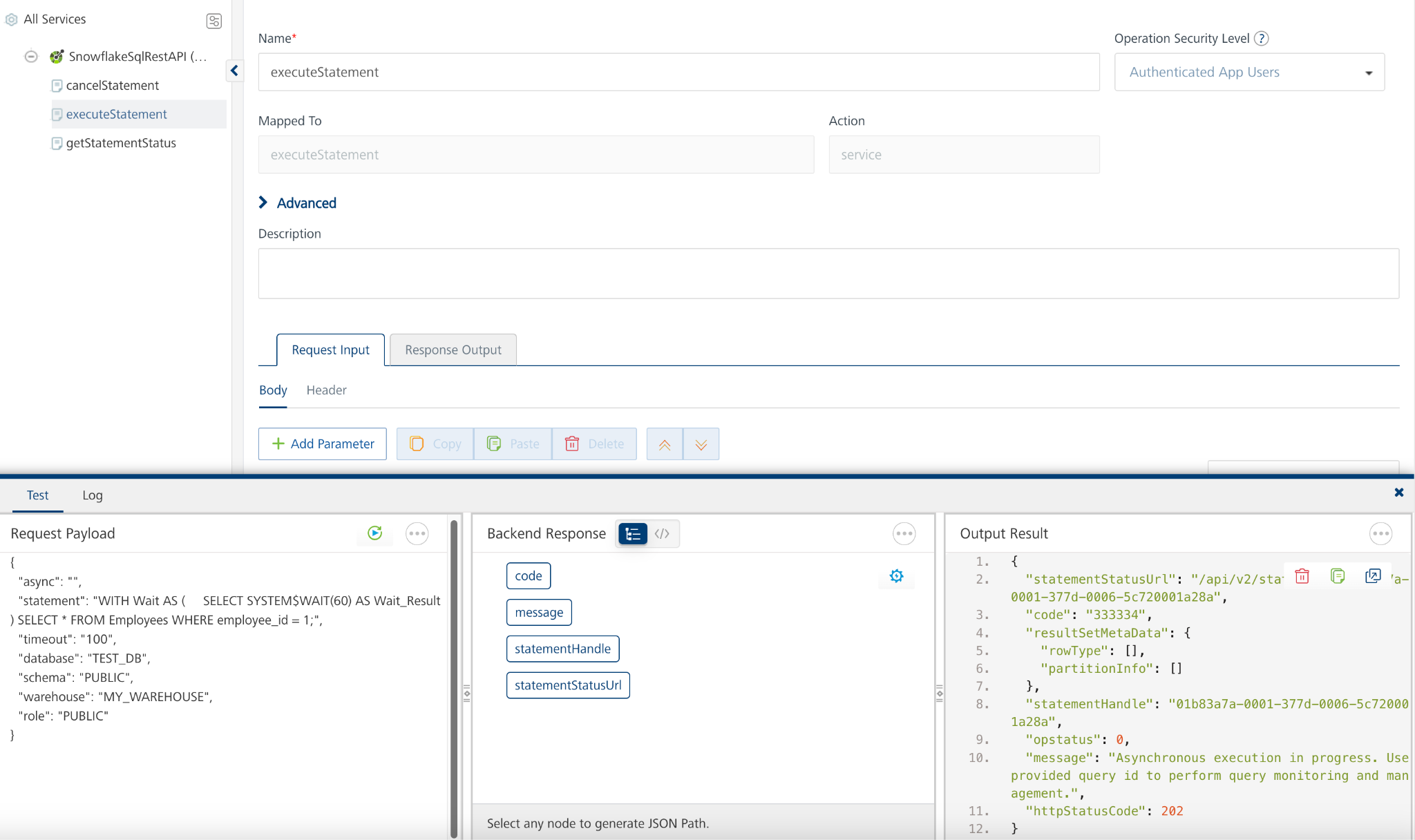
)

SELECT \*

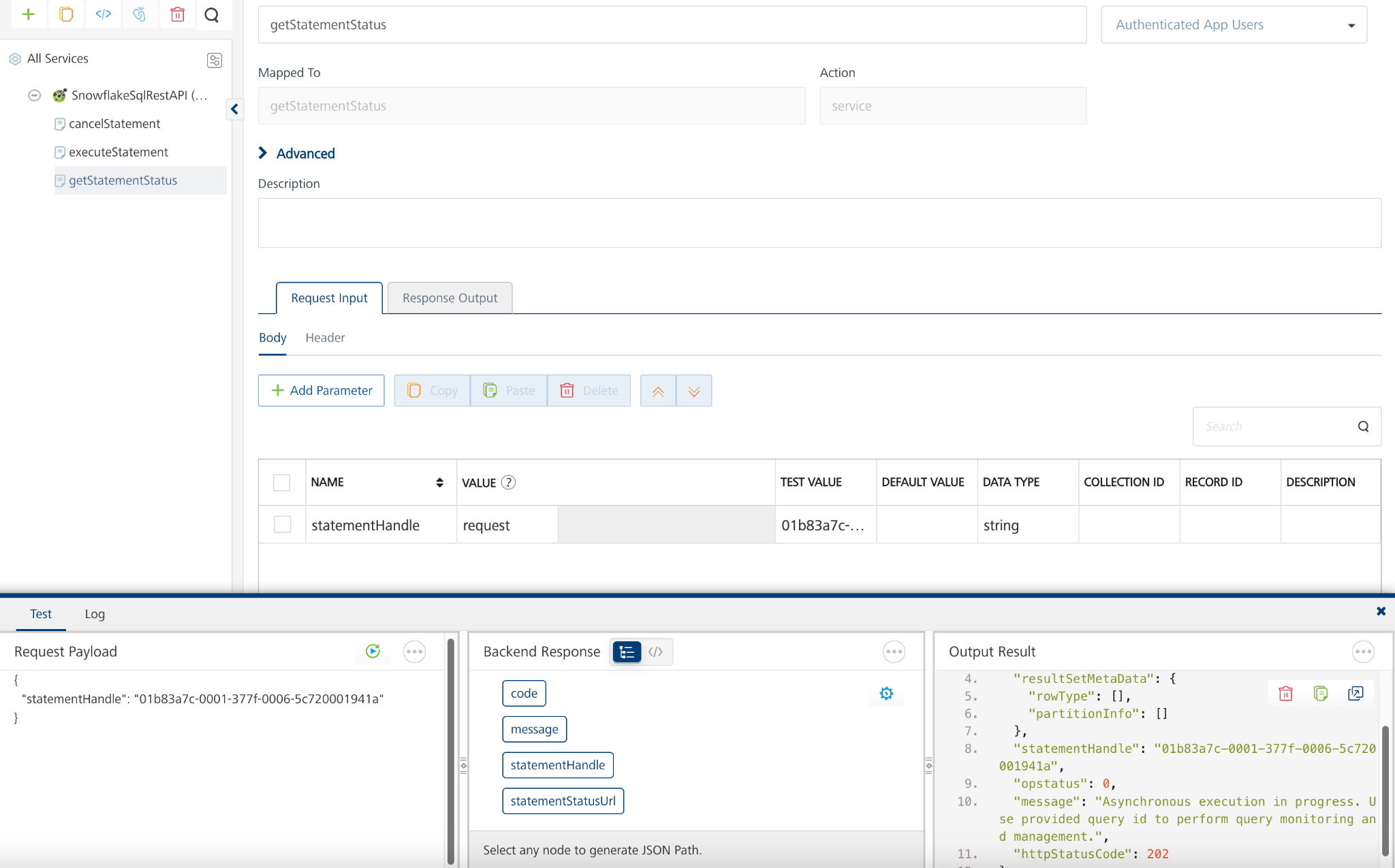
FROM Employees

WHERE employee\_id = 1;

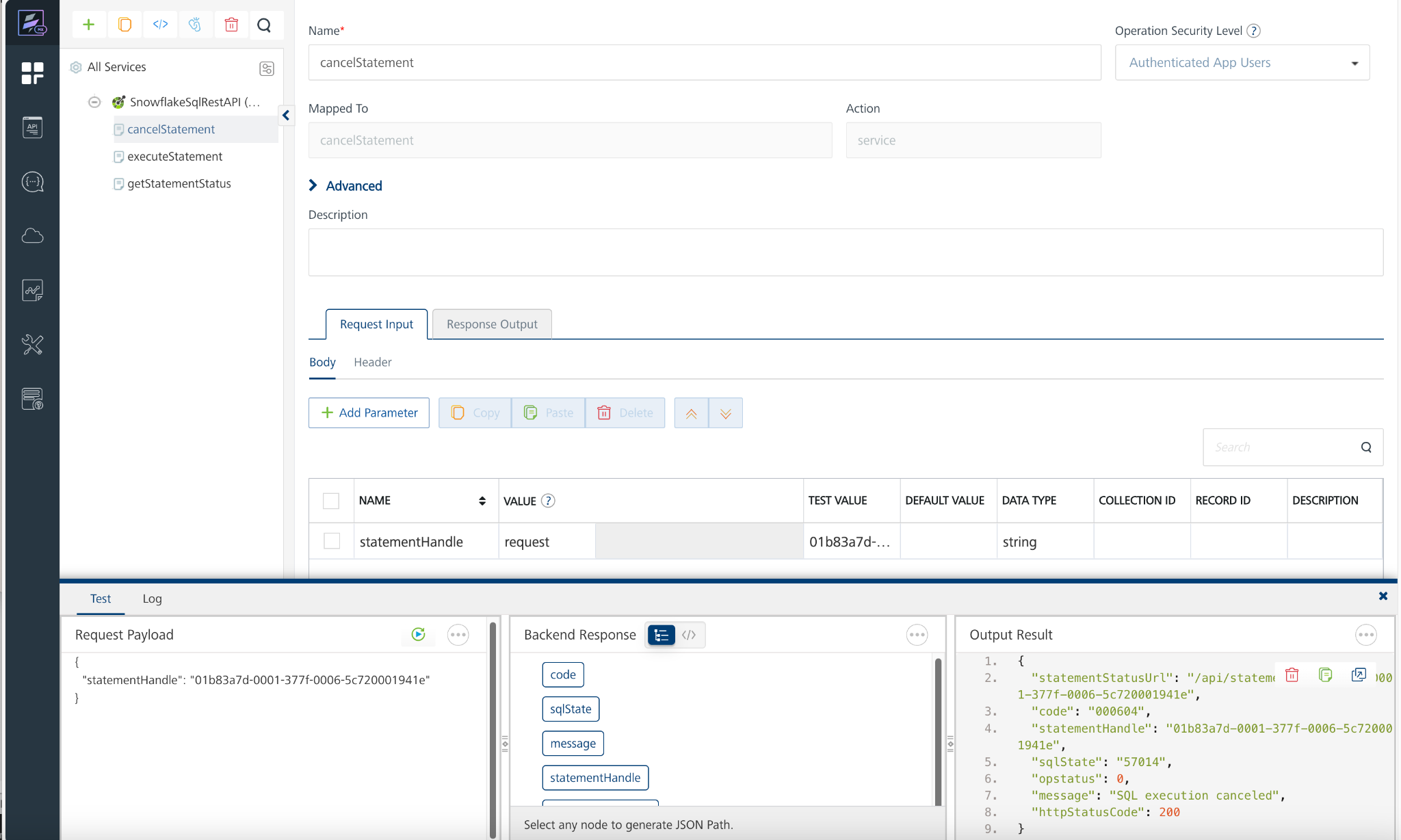
This query utilizes the SYSTEM$WAIT function to pause for 60 seconds, ensuring the query runs longer than the 45-second threshold. Once this query is executed, retrieve the statementHandle from the response. You can then use this statementHandle to:



**2. Check the status** of the statement using GET /api/v2/statements/{statementHandle}. In the above screenshot we got statement handle by using that statement handle we can check status of statement with in timeout what we have provided.



1. **Cancel the execution** if desired, using POST /api/v2/statements/{statementHandle}/cancel. In the above execute statement screenshot we got statement handle by using that statement handle we can cancel of statement within timeout what we have provided.



**3. Revision History**

App version 1.0.0:

1. **Known Issues**

-

1. **Limitations**

-