

**DataHub Framework &   
Technical Documentation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Charter Version History** | | | |
| **Version** | **Author** | **Version Date** | **Comments** |
| 1.0 | Jason Miller | 05/28/19 | Initial Draft |
| 1.1 | Jason Miller | 06/10/19 | Edits; added DataHub Components section |
| 1.2 | Jason Miller /Casey Colby | 09/04/19 | Final user documentation and post-development commentary |

Contents

[Overview 3](#_Toc18914098)

[Architecture 4](#_Toc18914099)

[Approach 5](#_Toc18914100)

[Security and Access 6](#_Toc18914101)

[DataHub Components 6](#_Toc18914102)

[User Guidance for Requesting Developer 8](#_Toc18914103)

[Guidelines for Database Review 8](#_Toc18914104)

[Guidelines for Creating SQL Queries 11](#_Toc18914105)

[Guidelines for an Apigee/DataHub Request Format 11](#_Toc18914106)

[DataHub Response Success and Error Samples 16](#_Toc18914107)

[Limits: Best Practices and Policies 17](#_Toc18914108)

## Overview

NU currently has multiple relational databases in the ecosphere of tools within NUIT. However, it is difficult to access the data stored within these relational databases by other NU systems.

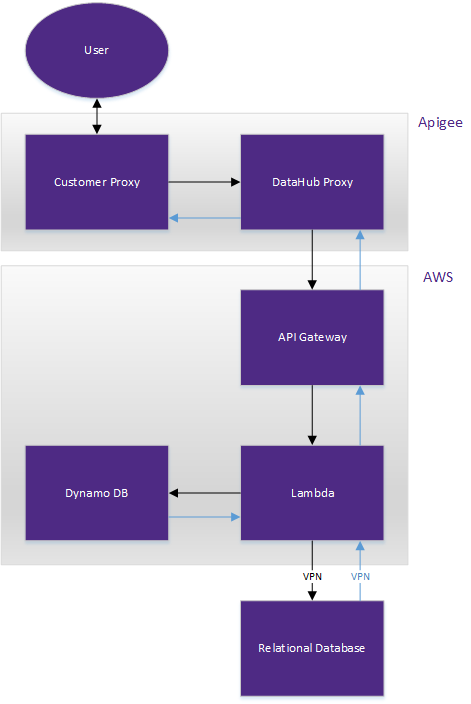
For example, **InfoEd** is a relational database containing valuable research grant application data that the Research Portal (which tracks active research projects) could utilize if that data were easier to access. The value of the data is lost when it cannot be called by other systems.

What is needed is a middleware translator that can receive an API call from any system, translate that call into an SQL query against a relational database and retrieve the data, and then convert that data into JSON in the response.

We are calling that middleware translator “DataHub”.

The purpose of DataHub is to expose the data in a relational database as RESTful endpoints that can be easily accessible to other systems. The following diagram represents the high-level sequence.

## Architecture



## Approach

**Apigee** will be used as a proxy for all RESTful endpoints; no consumer will hit DataHub directly.

The **customer API proxy** will call another API proxy called the **DataHub proxy**. All customer proxies, no matter which query is being called, will call the DataHub proxy; it is a many-to-one relationship.

**NOTE:** A customer proxy is required in order to standardize access and approval over which users are allowed to run which queries.

The customer proxy needs to contain all data that is required by the DataHub proxy. Potential data might include:

* **Query key** (which should only be known by the owner of the query and not made available to other groups)
* **Throttling limit** (optional policy applied to customer proxy)

**NOTE**: the customer proxy will contain a flow for each query.

All predefined parameters will be passed as key/value pairs within an HTTPS GET with a specific format TBD.

The DataHub proxy will pass its data into the core of DataHub, which is based in AWS, via the AWS API Gateway. From there, a lambda function will use the key to select the appropriate SQL query stored in Dynamo DB.

In order for the SQL queries to run correctly, different relational databases will require unique configurations (Oracle, MySQL, etc.). DataHub will include the appropriate code for each supported environment type.

Once the SQL query, the predefined parameters, and the code for the environment type have been compiled, the query will run using a username specifically created for DataHub within the relational database.

When the payload is returned, DataHub will convert from the SQL response format into JSON and return the JSON response back through the DataHub proxy and into the customer proxy (both via Apigee).

**NOTE:** Because all customer proxies will run through Apigee, additional filters and sorts could be applied to the returned JSON results using Apigee functionality available to the customer proxy.

## Security and Access

For **DataHub V1**, two keys will be used. One key will be included in the customer proxy that is used to gain access to the endpoint for each query. A second API key will be used between the DataHub proxy and the API Gateway so that only requests coming from Apigee can access the initial lambda function.

Security and access will be controlled per endpoint within Apigee using only API keys, allowing generic access to the endpoint and whatever data that endpoint returns. (If you have the key, you get the full data set that is returned.)

**NOTE:** The results can be filtered by the application calling for the data, but DataHub will not filter the data before returning it via the endpoint. In the future, parameter-based permissions could be added so the returned data is limited to the data allowed by the permission.

**NOTE:** DataHub is not responsible for data governance approval; groups who will consume the data from a relational databases via DataHub must gain approval from the appropriate data steward as part of the Apigee consumer-request process before making a request to configure a query.

## DataHub Components

DataHub is comprised of six components:

1. **Customer proxy API** within Apigee, which is comprised of:
   * API key associated with specific query endpoint
   * Predefined parameters for the SQL query (if needed)
   * Acceptance header information (XML or JSON)
   * Throttling limit (optional)
2. **DataHub proxy API** within Apigee, which is comprised of:
   * API query key, passed in from the customer proxy
   * Predefined parameters for the SQL query, passed in from the customer proxy
   * API key to pass through the API Gateway
   * Response JSON or XML
3. **API Gateway configurations**
   * This will include a policy to restrict IP address that do not come from Apigee.
   * The gateway will be a single endpoint for all traffic from the DataHub proxy.
   * A gateway endpoint will exist for each environment: DEV, QA, and PROD.
   * The gateway will check for the correct key coming in via the DataHub proxy.
4. **Lambda function,** currently one function (this could change during future development cycles) which includes the following sections:
   * Using the API query key, look up the correct SQL query in the Dynamo DB tables
   * Compile the static SQL (from DB tables), the predefined parameters (passed in via customer proxy), and the environment code (included in the lambda code)
   * Execute the resulting SQL query
   * Convert the SQL results into JSON
   * Return the JSON results to Apigee (JSON results will all use string data types)
5. **Two Dynamo DB tables**:
   * Query table containing: API query key, SQL statement, connection ID
   * Connection table containing: connection ID, username, password, connection string, DB type, driver information
6. **Jenkins pipeline** for continuous deployment to three environments:
   * Development (DEV)
   * Quality Assurance (QA)
   * Production (PROD)

## User Guidance for Requesting Developer

This section is intended as a guide for the developer who is requesting to pass an API through DataHub.

There are six overarching tasks for the requesting developer. The requesting developer will:

1. Review database for possible issues.
2. Create an SQL query for the database where the data is stored.
3. Create a customer proxy (if does not exist) and endpoint in Apigee.
4. Request a DataHub entry be created by the ADO team.
5. Receive the API key from the ADO team to run the API.
6. Test the return response.

### Guidelines for Database Review

#### Supported Target Databases

|  |  |
| --- | --- |
| **Database** | **Supported Versions** |
| Oracle Database | Oracle Database 10.2 or greater |

#### DataHub Constraints

|  |  |
| --- | --- |
| **Resource** | **Limit** |
| Customer Request Timeout | 18 Seconds |
| Response Payload | 6MB |
| DB Access | Read only |
|  |  |

For Oracle databases, be aware of the following:

* **Duplicate column names**: If multiple columns have the same name, only the later column is returned. Use aliases or another strategy in your query to mitigate duplicate column names.
* **Empty/missing values**: If a row does not specify a value for a column, it will appear as ‘null’ in the DataHub JSON object.
* **Date formats**:

#### The following database row …

|  |  |
| --- | --- |
| START\_DATE | END\_DATE |
| 01-JAN-16 | 31-AUG-16 |

#### … would return the following JSON from DataHub:

{

"results": [

{

“START\_DATE”: “2016-01-01T00:00:00.000Z”,

"END\_DATE": “2016-08-31T00:00:00.000Z”

}

]

}

* **Large-Object (LOB) Columns**: (e.g. XML Forms) Newline characters \n and carriage returns \r will be substituted in the json response. Quotations will be escaped with a \ in the json response.

**Example: LOB with Newline Characters, Carriage Returns, and Quotations**

#### The following database row …

|  |  |
| --- | --- |
| STUDENT\_ID | XMLFORM\_01 |
| xyz123 | <Form01>  <entry1>  <section1>  <question0>000123</question0>  <question1>  <row id="1" guid="undefined" key-childname="johnsmith">  <question2>northwestern</question2>  </row>  </question1>  </section1>  </entry1>  </Form01> |

#### … would return the following JSON from DataHub:

{

"results": [

{

“STUDENT\_ID”: “xyz123”,

"XMLFORM\_01": "<Form01>\r\n <entry1>\r\n <section1>\r\n <question0>000123</question0>\r\n <question1>\r\n <row id=\"1\" guid=\"undefined\" key-childname=\"johnsmith\">\r\n <question2>northwestern</question2>\r\n </row>\r\n </question1>\r\n </section1>\r\n </entry1>\r\n</Form01>"

}

]

}

### Guidelines for Creating SQL Queries

* Do not put any sensitive data in the parameters.
* Denote parameters with a colon.
  + e.g. SELECT \* FROM people WHERE fname=:firstname
  + specifies that a parameter will be supplied for the value of firstname
* You can only parameterize the data values in the query (as opposed to the object names like the table, columns, etc.).
* In the example below, “people” is a table object name and “fname” is column object name.
  + Acceptable: SELECT \* FROM people WHERE fname=:firstname
  + Not Allowed: SELECT \* FROM :table\_name
  + Not Allowed: SELECT :fieldtoreturn FROM people

### Guidelines for an Apigee/DataHub Request Format

#### Prerequisites

The requesting develop is responsible for completing the following before making a DataHub request:

1. Create a new **read-only datahub user** for the database.
2. Create an **Apigee Customer Proxy.** (See more instruction below.)
3. **Firewall Request** for connecting to the database from DataHub:
   * Dev/Test: request access for source address **10.28.192.160/27**
   * Prod: request access for source address **10.28.196.160/27**

#### DataHub Requests

Once the steps above are complete, the requesting developer should supply the following to the DataHub team:

1. Database Username
2. Database Password
3. Database Connection String
4. Parameterized SQL query (Provide a **single string**—i.e. *do not include newlines/carriage returns in the query unless you accept their replacement with \n \r characters in the final query)*

#### Apigee URL Overview

Apigee URLs consist of four parts:

* The Apigee environment, which will be (marked in red)
* The Apigee customer proxy (marked in green)
* The Apigee flow (marked in blue)
* Any parameters, *optional* (marked in purple)

**Format: Apigee URL with No Parameters**

https://northwestern-{{apigee-env}}.apigee.net/customer-proxy-name/flow-name

Any parameter values are required in the query parameters of the request URL as name-value pairs in the query parameters at the end of the URL, where the name in the name-value pair matches the parameter placeholder in the SQL statement

**Format: Apigee URL with Parameters**

https://northwestern-{{apigee-env}}.apigee.net/customer-proxy-name/flow-name  
?parameter1=value1&parameter2=value2&parameter3=value3

**Example: Query and Apigee URL**

SELECT \* FROM people WHERE fname=:firstname AND lname=:lastname

<https://northwestern-prod.apigee.net/sample-customer-proxy/peoplebyname>  
?firstname=john&lastname=smith

#### Customer Proxy Starter Agreement

The DataHub team is providing a quick-start customer proxy .zip that will make it much quicker for you to set up your customer proxy and get started with datahub.

We are here for questions!

**However by using this starter, you acknowledge that *you* are responsible for your customer proxy including:**

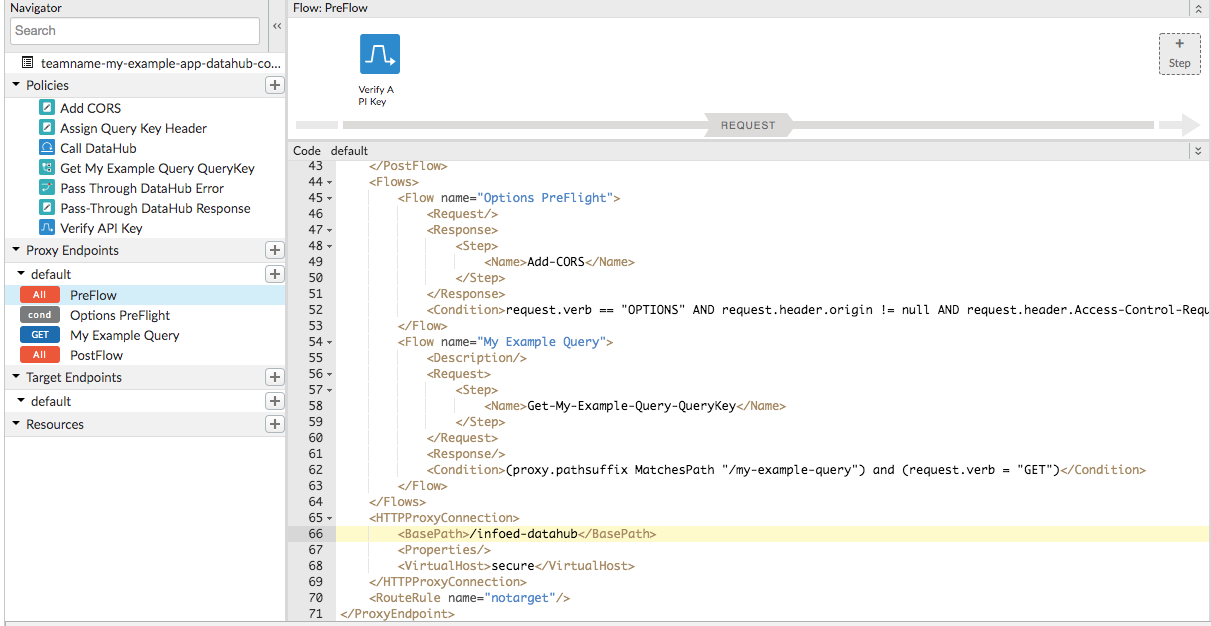
* **Enhancements, Changes, Updates**
* **Debugging, Troubleshooting, Errors**
* **Generally understanding how it works**

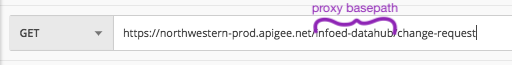
If you would feel better equipped to own your customer proxy by building it yourself, we can provide step-by-step instructions for building a customer proxy from scratch.

#### Customer Proxy Creation

Complete **only** the following steps from the [API Service Registry Documentation](https://apiserviceregistry.northwestern.edu/producer/steps):

* Create API Proxy **from bundle** (do not follow the Reverse Proxy instructions on the service registry documentation)
  + Proxy Name: {{teamname}}-{{appname}}-datahub-connector
  + Bundle: datahub-customer-starter.zip
* Create API Products: DEV/TEST, PROD
* Submit Form: Register New API (Also request in your ticket that the products be added to your team’s Apigee testing app so you can test/call the service, or request an app be created if you don’t have one.)
* If you need help with Apigee basics in this section (creating a proxy/products or using your apikey), [contact the DMA Integration Team](https://apiserviceregistry.northwestern.edu/Contact)
* In the Proxy Endpoints section, locate the <HTTPProxyConnection> and update the <BasePath> for calling your customer proxy (typically would be of the format /appname-datahub)
* This will be in the url to call your proxy, after http://northwestern-{{env}}.apigee.net (e.g. /infoed-datahub)

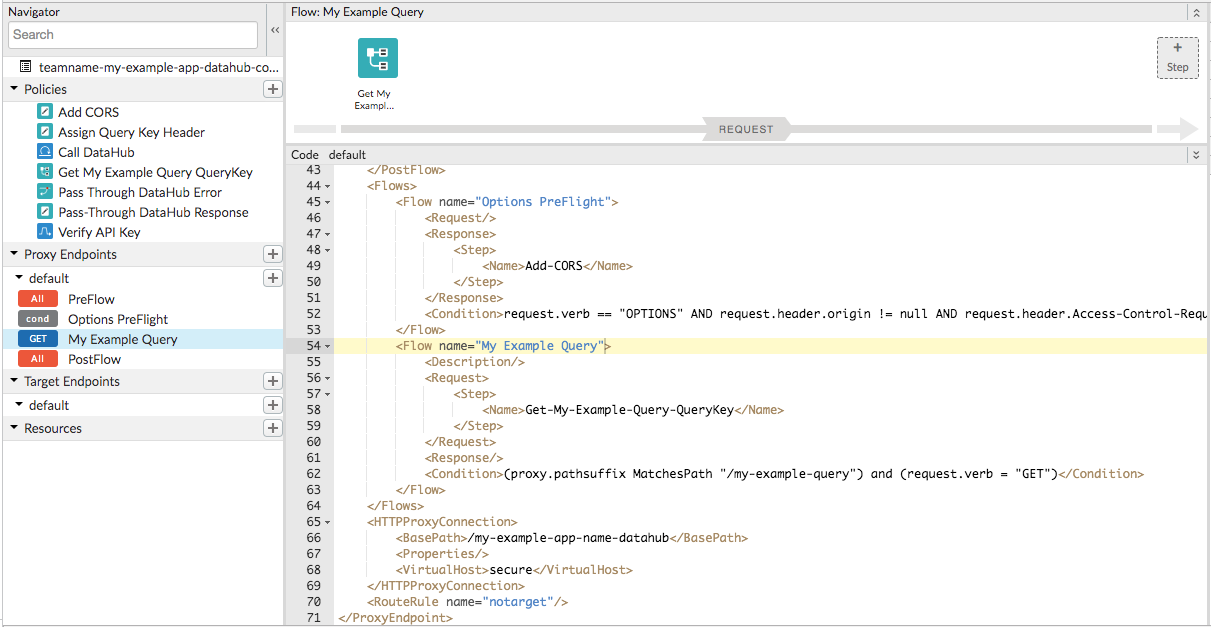


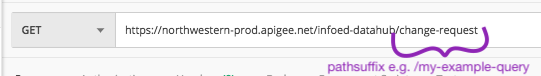


#### Final Apigee Configuration

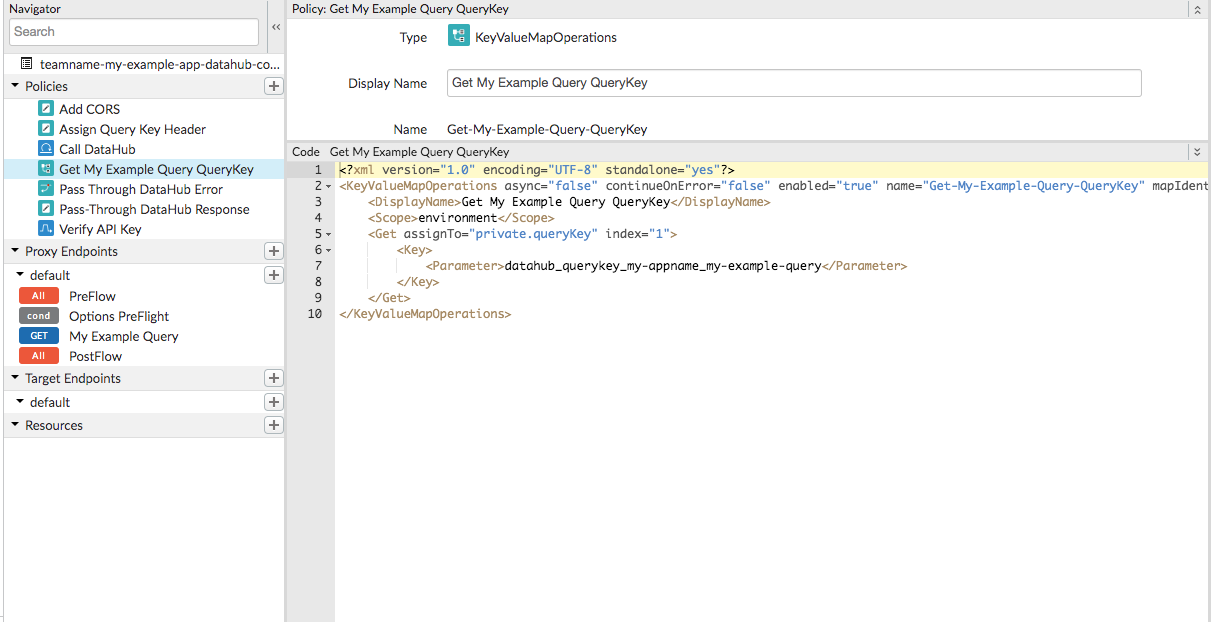
Once you have set up a new query with the DataHub team and were provided a query key, there are several steps of final configuration within Apigee.

1. Create a new flow under the Proxy Endpoints section like My Example Query below. (Replace the 2 instances of *my example query* with your query name.)



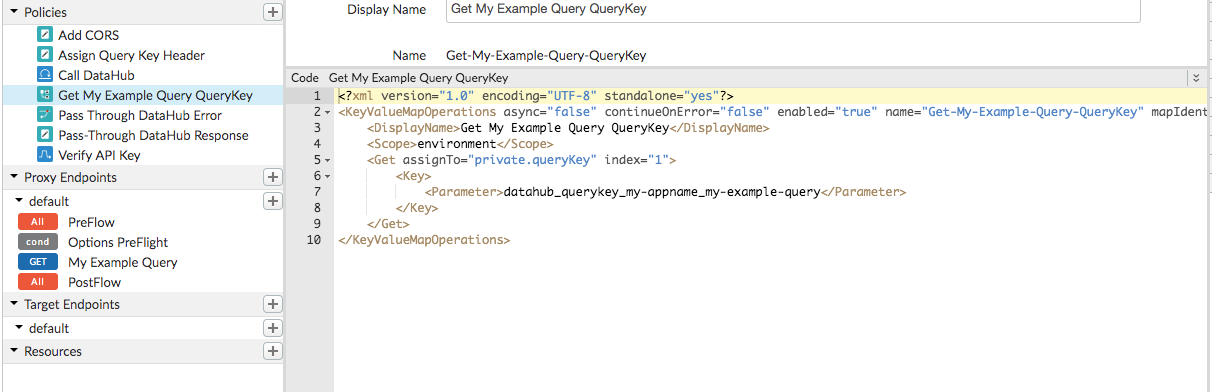


1. Create a new KVM policy to look up the query key under Policies section like Get My Example Query QueryKey below. (Replace *my appname* and *my example query* with your own.)

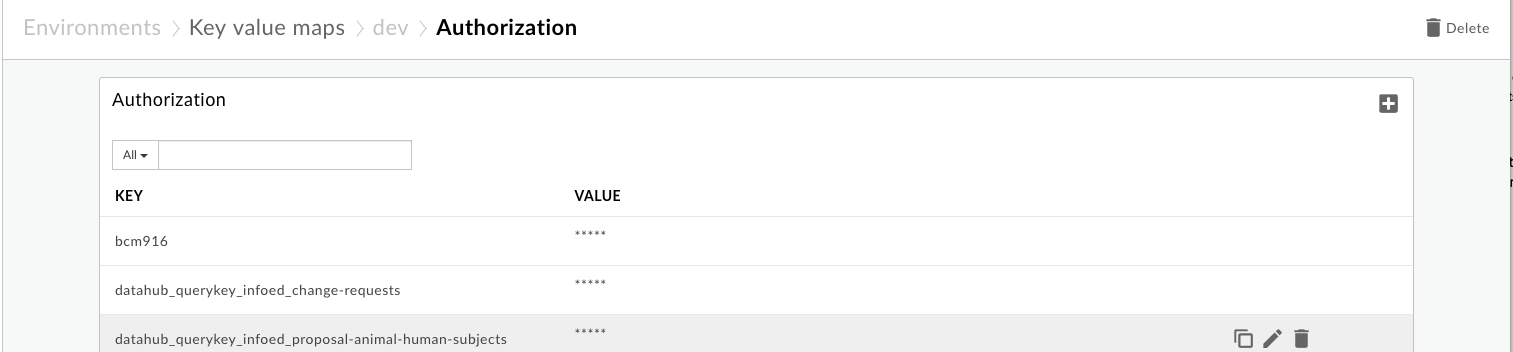


1. Add the query key to the Authorization Key Value Map in Apigee (per environment) with the name specified in the ‘Get [QueryName] QueryKey’ policy.

(A) Get parameter specified in QueryKey policy



(B) In each environment’s Authorization KVM



(C) Add the Query Key value with the name specified in figure A

1. Clean up the My Example Query flow and KVM policy once you no longer need them.

### DataHub Response Success and Error Samples

#### Success

A successful DataHub request returns an application/json object with an array of objects in the “results” property.

Each object in the datahub Results array corresponds to one row returned from the target database, and the column headers of the database rows map to the json object keys.

*Sample Rows Returned from Target*

|  |  |  |
| --- | --- | --- |
| STUDENT | FNAME | LNAME |
| yes | John | Smith |
| no | Jane | Doe |

*Sample JSON Returned from DataHub*

{

"results": [

{

"STUDENT": "yes",

"FNAME": "John",

"LNAME": "Smith"

},

{

"STUDENT": "no",

"FNAME": "Jane",

"LNAME": "Doe"

},

]

}

#### Error

An error datahub request returns an application/json object with an error detail in the “error” property.

The DataHub Open API Spec lists specific error types/codes .

Sample error returned from DataHub

{

"error": {

"message": "Query not found.",

"code": 404,

"type": "ResourceNotFoundError",

"category": "CustomerConfigurationError"

}

}

### Limits: Best Practices and Policies

DataHub payload and timeout limits are hard limits within which we have to work. DataHub best practices and policies for working within these are below.

#### Large Queries

Queries which exceed the DataHub response size limitation return a 413 error:

{

"error": {

"message": "Response Payload exceeds 6MB maximum. Configure as Large query.",

"code": 413,

"type": "ResponseTooLargeError",

"category": "CustomerConfigurationError"

}

}

Queries which exceed the limit may be configured to return instead a presigned url in the response body to retrieve the response which will expire in 30 seconds.

However additional evaluation/approval is required to configure a large query as it must be determined that your query follows best practices (e.g. the large query is efficient/justified and would not be better suited to a series of smaller queries) and performance/cost impacts must be considered.

**IMPORTANT NOTE:** You are still constrained by the DataHub Timeout limit to return data on a large query!

{

"url": "https://datahub-large-response-prod.s3.us-east-2.amazonaws.com/xyz123"

}

#### 504 Gateway Timeout

It is recommended you add logic in your application to handle a 504 Gateway Timeout error, to account for the occasion when DataHub can’t respond in a timely manner.

* Retry the request one subsequent time
* Failing that, gracefully return a friendly error and handle the user experience rather than returning default DataHub error

If your query runs longer than the timeout limits allow you will need to make changes to your query.