GraphQL API + Qlik Integration

Technical Brief

Last Updated: 9/9/2020
Verified using Qlik Sense Desktop June 2020

Table of Content

Introduction

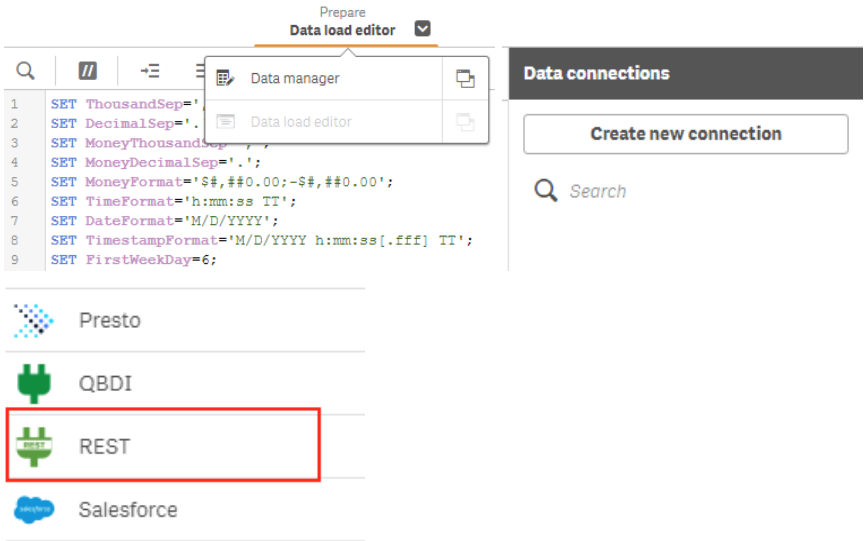
The Pluralsight GraphQL API can be utilized within Qlik by the use of SQL-Language code based queries being set up within the application. The method for pulling the data is easy to replicate for all graphql objects, though you must review the schema documentation if you want to know what Field data is available to pull. For a quick start, review the resources section below. For a more in depth explanation, read on.

Prerequisites

- Experience using QlikView or Qlik Sense Desktop (both scripting and visualizations)
- You will need the following prior to following along:
  - Qlik Sense Desktop
  - Pluralsight PaaS API Key.

Data Connection Settings

To begin, you will want to open Qlik and go to **Create new app** >**Name of my app** > **Data Load Editor->Create new connection->REST**. See screenshots below.

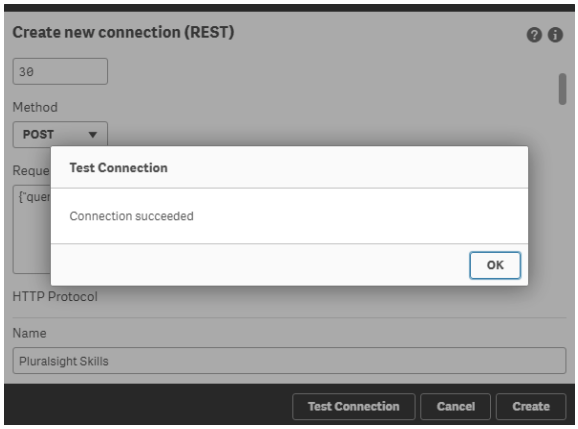


Next fill out the new connection fields with the following values:

Category	Title	Value
Request	URL	https://paas-api.pluralsight.com/graphql

	Timeout	30
	Method	POST
	Request Body	{"query":"query { users{ nodes { id, email} } }"}
	HTTP Protocol	1.1
	Request/Response Body Encoding	UTF-8
	Add 'Expect: 100-continue' header	Checked
Data Options	Auto detect Response type	Checked
	Check response type during 'Test connection'	Checked
	Key generation strategy	Sequence ID
Authentication	Authentication Schema	Anonymous
	Skip Server Certificate Validation	Checked
	User Certificate	No
Additional Request Parameters	Add Missing Query Parameters to Final Request	Checked
Query Headers	Authorization	Bearer <Paste Skills PaaS API Key Here>
	Content-Type	application/json
Pagination	Pagination type	None
Security	Allow response headers	Unchecked
	Allow HTTPS only	checked
	Redirect URL Whitelist	<Leave Blank>
Proxy	Use Proxy	Unchecked
	Connection Name	Pluralsight Skills

After entering everything correctly, Test Connection and you will receive a Connection Succeeded as shown below:

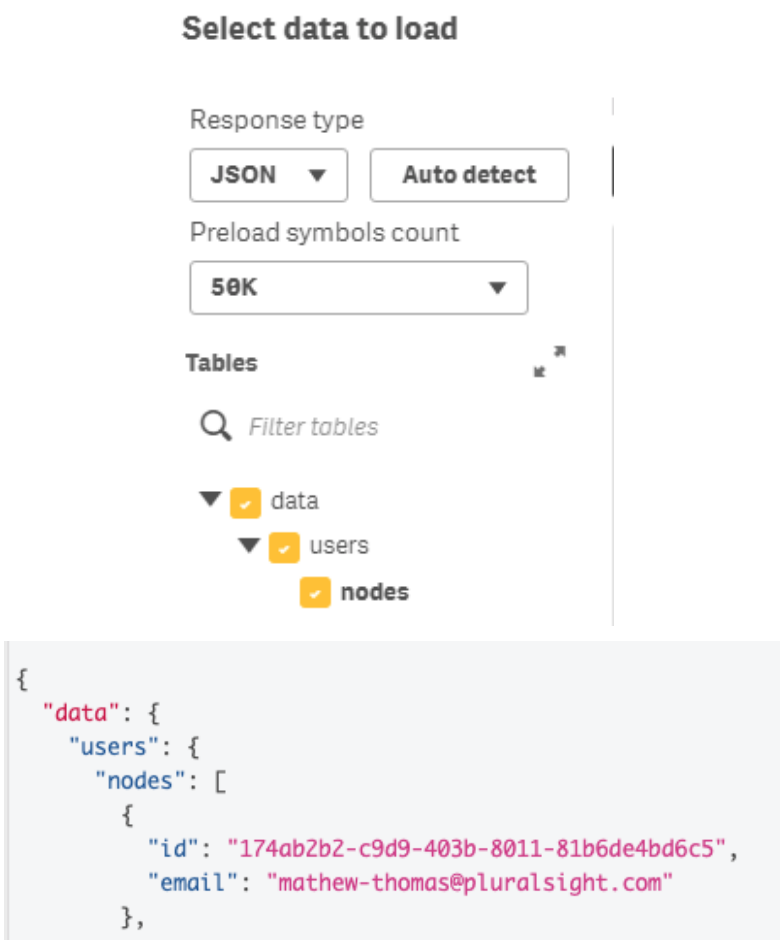


Basic Query Structure

To begin, you will need to establish a connection by writing the following line.

```
LIB CONNECT TO 'Pluralsight Skills';
```

The whole query will need to be written into a new tab. The basic structure of the query is outlined below. The first place to observe is the fact that each query will require 3 select statements to successfully query the results of the data. The reason for this is GraphQL returns three nested objects before you arrive at the array of data. Below shows the path to reach the data which is data->users->nodes.




```
{
  "data": {
    "users": {
      "nodes": [
        {
          "id": "174ab2b2-c9d9-403b-8011-81b6de4bd6c5",
          "email": "mathew-thomas@pluralsight.com"
        },
```

The next main section represents the fields you want to return from GraphQL. In the below example, Qlik is selecting the following fields:

- "id",
- "email",
- "firstName",
- "lastName",
- "startedOn",
- "note",
- "isOnAccount",
- "removedOn",
- "ssoEnabled"

You will be able to find the field names for each schema (tables) by reviewing the documentation: https://dev-portal.pluralsight.com/schema-docs/query.doc.html.
The last section wraps the json call using the WITH CONNECTION which overrides the data connection parameters. The WITH CONNECTION is useful

when needing pagination which is defined in the pagination section below. The WITH CONNECTION is wrapping the following parameters:

- URL - defines the GraphQL end point
- QUERY - defines the response type (JSON)
- HTTPHEADERS - GraphQL requires two headers a) Authorization b) Content-Type
- Body - Query String that needs to be sent to GraphQL to return the data. To learn what query to sent to GraphQL, please leverage the following resource:
    - GraphQL Playground

Please note, you will need to replace **API KEY** with your actual API token that you can generate on the Developer Portal found in the Resources section. You can then reuse this section for all other queries against the Pluralsight GraphQL API.

```
SQL SELECT
      "__KEY_data",
      (SELECT
            "__KEY_users",
            "__FK_users",
            (SELECT
                  "id",
                  "email",
                 "firstName",
                  "lastName",
                 "startedOn",
                 "note",
                 "isOnAccount",
                 "removedOn",
                 "ssoEnabled"
            FROM "nodes" FK "__FK_nodes")
      FROM "users" PK "__KEY_users" FK "__FK_users")
FROM JSON (wrap off) "data" PK "__KEY_data"
WITH CONNECTION (
      Url "https://paas-api.pluralsight.com/graphql",
      QUERY "type" "JSON",
      HTTPHEADER "Authorization" "Bearer <API KEY>",
      HTTPHEADER "Content-Type" "application/json",
      BODY "{""query"":""query{ users { nodes { id, email,
firstName, lastName, startedOn, note, isOnAccount, removedOn,
ssoEnabled } } }"" }"
```

```
);
```

Basic Sub Query Structure

Pagination Structure

This section focuses on paging through the query. By default, the GraphQL will return the first 100 rows unless specified the cursor-based pagination. This type of pagination allows you to specify how many pieces of data you want to read, and where you want to start reading from. Each data entry has a cursor, which is an opaque string identifying the entry's place in the greater dataset. Queries that offer pagination will have the following two parameters:

- first: Specifies how many records to fetch per query (limited to 10000 per query).
- after: Used to request the next result set from the query. Your query will return the value PageInfo.endCursor, and you should put that value here for the next query. (has a rate-limit of 100 requests per minute)

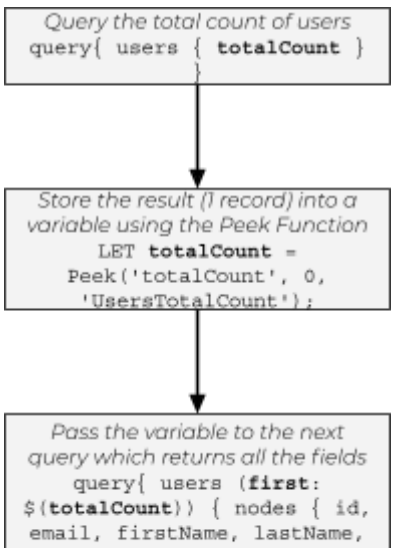
The Qlik Rest Connector does support pagination however it doesn't support cursor-based. There are two ways to solve for pagination: query-size pagination or recursive pagination. Each comes with their own pros and cons. Below describes both:

Query-Size Pagination Method

The query-size pagination method leverages the first parameter to be inserted into the query string inside the body. As an example, let's use the basic query:

```
"{""query"":""query{ users { nodes { id, email, firstName,
lastName, startedOn, note, isOnAccount, removedOn, ssoEnabled } }
}
```

By default every query sent to GraphQL has additional properties that can be added with the query. For this method, it requires the following two properties which are **totalCount** and **first**. **TotalCount** will return the total rows of data that can be fetched and **first** specify the number of records to fetch per query. Therefore, by leveraging these two properties, one can write a query that fetches the **first totalCount** of records. To the right illustrates a flow diagram on the logic.



With this logic, the query mentioned above will need to be modified to include the ***first*** keyword and the variable name (e.g. totalcount).

```
"{""query"":""query{ users (first: $(totalcount)) { nodes { id,
email, firstName, lastName, startedOn, note, isOnAccount,
removedOn, ssoEnabled } } }"" }"
```

As mentioned above, this approach comes with some pros and cons which are listed below:

Pros	Cons
• Easier to implement • Easier to modify / update when required	• Limited to 10,000 rows • Requires temporary tables to fetch the total count and then drop it

Finally, below is a full sample script on how this method will work. The full sample script is to pull in Users from GraphQL:

```
//Get total count of records for users table
UsersTotalCount:
SQL SELECT
      "__KEY_data",
      (SELECT
         "totalCount"
      FROM "users" PK "__KEY_users" FK "__FK_users")
FROM JSON (wrap off) "data" PK "__KEY_data"
WITH CONNECTION (
      Url "https://paas-api.pluralsight.com/graphql",
      QUERY "type" "JSON",
      HTTPHEADER "Authorization" "Bearer <API KEY>",
      HTTPHEADER "Content-Type" "application/json",
      BODY "{""query"":""query{ users { totalCount } }"" }"
);

//Store the total count into a variable
LET totalCount = Peek('totalCount', 0, 'UsersTotalCount');

//Discard UsersTotalCount table as it's not needed anymore
DROP TABLE UsersTotalCount;
```

```
//Pass the total count inside the Body's query by using the filter
first
Users:
SQL SELECT
      "__KEY_data",
      (SELECT
            "__KEY_users",
            "__FK_users",
            (SELECT
                  "id",
                  "email",
                  "firstName",
                  "lastName",
                  "startedOn",
                  "note",
                  "isOnAccount",
                  "removedOn",
                  "ssoEnabled",
                  "__FK_nodes"
            FROM "nodes" FK "__FK_nodes")
      FROM "users" PK "__KEY_users" FK "__FK_users")
FROM JSON (wrap off) "data" PK "__KEY_data"
WITH CONNECTION (
      Url "https://paas-api.pluralsight.com/graphql",
      QUERY "type" "JSON",
      HTTPHEADER "Authorization" "Bearer <API KEY>",
      HTTPHEADER "Content-Type" "application/json",
      BODY "{""query"":""query{ users (first: $(totalCount)) {
nodes { id, email, firstName, lastName, startedOn, note,
isOnAccount, removedOn, ssoEnabled } } }"" }"
);
```

Recursive Pagination Method

The recursive pagination method leverages the parameter **after** as well as it requires a new object called **pageInfo**. The **after** parameter allows the query to request the next result set. Your query will return the value PageInfo.endCursor, and you should put that value here for the next query.

As an example, let's use this basic query:

```
"{""query"":""query{ users { nodes { id, email, firstName,
lastName, startedOn, note, isOnAccount, removedOn, ssoEnabled } }
```

```
}"" }"
```

By default every query sent to GraphQL has additional properties that can be added with the query. For this method, it requires the following property **after** and a new object called **pageInfo** that contains two keys: **endCursor** and **hasNextPage** to be added. Using the property and object the query needs to be modify as follow:

```
"{""query"":""query { users (after: "") { pageInfo { endCursor,
hasNextPage } nodes { id, email} } }"" }"
```

In addition, since a new object is being introduced, the SQL statement needs to be updated to reflect this new table. Therefore, the new SQL statement will need to include a new Select statement to query the endCursor and hasNextPage value. Therefore the SQL statement will be updated with the following addition:

```
SQL SELECT
      "__KEY_data",
      (SELECT
          "__KEY_users",
          "__FK_users",
          (SELECT
              "endCursor",
              "hasNextPage",
              "__FK_pageInfo"
          FROM "pageInfo" FK "__FK_pageInfo"),
          (SELECT
              "id",
              "email",
              "__FK_nodes"
          FROM "nodes" FK "__FK_nodes")
      FROM "users" PK "__KEY_users" FK "__FK_users")
  FROM JSON (wrap off) "data" PK "__KEY_data"
  WITH CONNECTION (
      Url "https://paas-api.pluralsight.com/graphql",
      QUERY "type" "JSON",
      HTTPHEADER "Authorization" "Bearer <API Key>",
      HTTPHEADER "Content-Type" "application/json",
      BODY "{""query"":""query { users (after: "") { pageInfo {
endCursor, hasNextPage } nodes { id, email} } }"" }"
```

```
);
```

As stated earlier, GraphQL has a rate-limit of 100 requests per minute and since we are not using cursor-based pagination (using property **first**); each query will return 100 rows. So after each query, a check must be made to ensure we are within the rate-limit or an error will return. In order to solve this problem, three variable needs to be introduced:

```
LET vCounter = 0;//keeps track of how many query has been called
//keeps track of query time and if 1 min has elapsed
LET vNow = now();
LET vStartTime = num((vNow - MakeDate(1970,1,1))*86400);
LET vEndTime = num(((vNow +(1/(24*60)))-
MakeDate(1970,1,1))*86400);
```

After a query returns 100 results, these four variables can be used to check if the following two conditions have occurred:

1. Is the vCounter at 99
2. Is vStartTime (time when a query started - needs to be converted to epoch) is greater than vEndTime (a minute added to when a query started - needs to be converted to epoch).

If the two conditions above are met, then the script should sleep for 1 minute so that the rate-limiter resets. Below is the conditional logic:

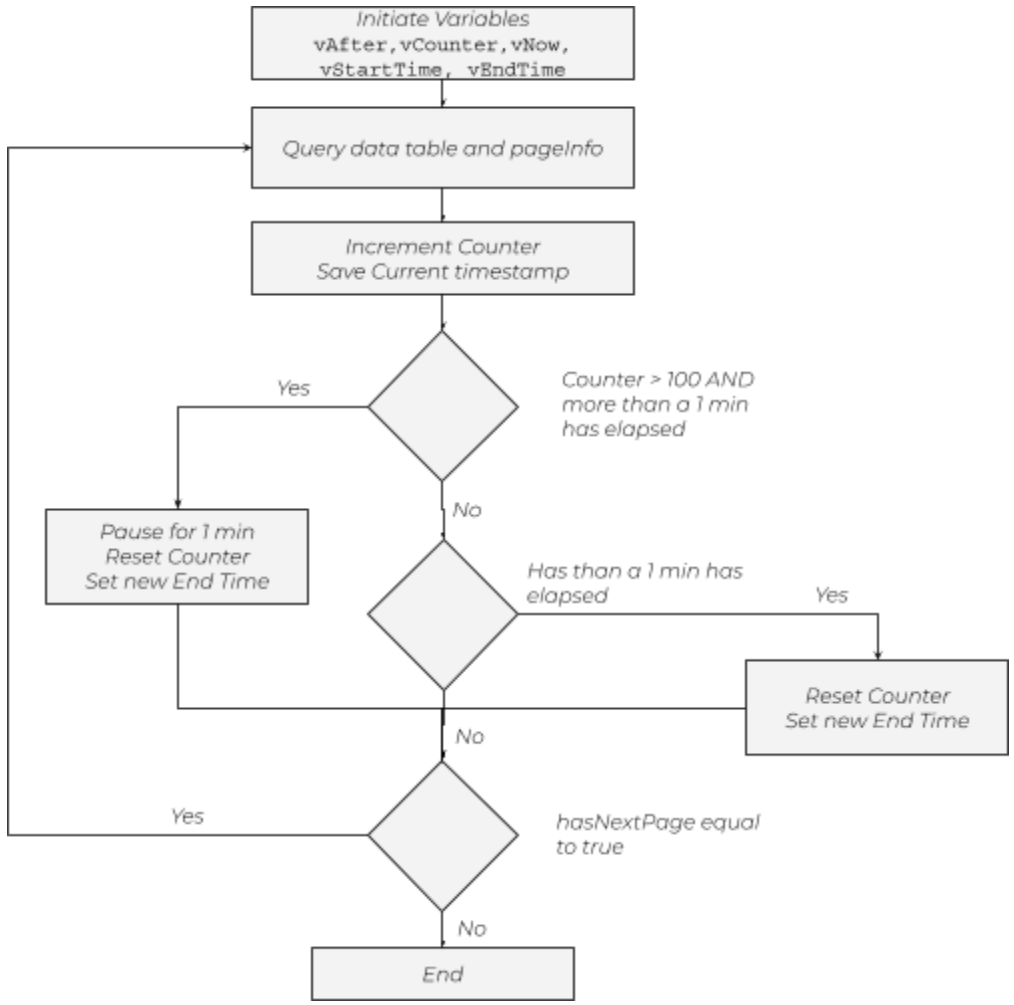
```
LET vCounter = $(vCounter) + 1;
LET vNow = now();
LET vStartTime = num((vNow - MakeDate(1970,1,1))*86400);

IF $(vCounter) = 99 AND $(vStartTime) > $(vEndTime) THEN
      Sleep 60000;
    LET vCounter = 0;
    LET vEndTime = num(((vNow +(1/(24*60)))-
MakeDate(1970,1,1))*86400);
ELSEIF $(vStartTime) > $(vEndTime) then
    LET vCounter = 0;
    LET vEndTime = num(((vNow +(1/(24*60)))-
MakeDate(1970,1,1))*86400);
END IF
```

As mentioned above, this approach comes with some pros and cons which are listed below:

Pros	Cons
• Unlimited number of rows to ingest • Takes into account the rate-limiter	• Complex coding, therefore more maintenance to up-keep • Requires multiple calls to GraphQL which will take longer for the data to load.

Finally the visual diagram below outlines how the script comes together.




```
LET vAfter = ''; //Initiate vAfter to blank
LET vCounter = 0;//keeps track of how many query has been called
per min
LET vNow = now(); //Stores the time
LET vStartTime = num((vNow - MakeDate(1970,1,1))*86400);
LET vEndTime = num(((vNow +(1/(24*60)))-
MakeDate(1970,1,1))*86400);
```

```
Do
  RestConnectorMasterTable:
  SQL SELECT
      "__KEY_data",
      (SELECT
          "__KEY_users",
          "__FK_users",
          (SELECT
              "endCursor",
              "hasNextPage",
              "__FK_pageInfo"
          FROM "pageInfo" FK "__FK_pageInfo"),
          (SELECT
              "id",
              "email",
              "__FK_nodes"
          FROM "nodes" FK "__FK_nodes")
      FROM "users" PK "__KEY_users" FK "__FK_users")
  FROM JSON (wrap off) "data" PK "__KEY_data"
  WITH CONNECTION (
      Url "https://paas-api.pluralsight.com/graphql",
      QUERY "type" "JSON",
      HTTPHEADER "Authorization" "Bearer <API Key>",
      HTTPHEADER "Content-Type" "application/json",
      BODY "{""query"":""query { users (after: \""$(vAfter)\"") {
pageInfo { endCursor, hasNextPage } nodes { id, email} } }"" }"
  );

 [pageInfo]:
  LOAD
      [endCursor],
      [hasNextPage]
      //[__FK_pageInfo] AS [__KEY_users]
  RESIDENT RestConnectorMasterTable
  WHERE NOT IsNull([__FK_pageInfo]);

  LET vAfter = Peek('endCursor', 0, 'pageInfo');
  LET vHasNextPage = Peek('hasNextPage', 0, 'pageInfo');

  DROP TABLE pageInfo;
```

```
  [data]:
  LOAD [id],
       [email]
       //[__FK_nodes] AS [__KEY_users]
  RESIDENT RestConnectorMasterTable
  WHERE NOT IsNull([__FK_nodes]);

  DROP TABLE RestConnectorMasterTable;

  LET vCounter = $(vCounter) + 1;
  LET vNow = now();
  LET vStartTime = num((vNow - MakeDate(1970,1,1))*86400);

  IF $(vCounter) = 99 AND $(vStartTime) > $(vEndTime) THEN
      Sleep 60000;
    LET vCounter = 0;
    LET vEndTime = num(((vNow +(1/(24*60)))-
MakeDate(1970,1,1))*86400);
  ELSEIF $(vStartTime) > $(vEndTime) then
    LET vCounter = 0;
    LET vEndTime = num(((vNow +(1/(24*60)))-
MakeDate(1970,1,1))*86400);
  END IF

Loop while '$(vHasNextPage)' = 'True'
```

Conclusion

The Pluralsight GraphQL API can be utilized within Qlik by the use of SQL-Language code based queries being set up within the application. Qlik allows developers to combine data from multiple data sources which can help customers to enhance Pluralsight's GraphQL data. By combining both Pluralsight's GraphQL data with internal customer data, customers can create unique analytical views that are tailored to their objectives.

Resources

- GraphQL Documentation hosted on our Developer portal
  - Pluralsight Developer Portal: Home
- Playground for exploring & converting query to basic Qlik query
  - GraphQL Playground
- Qlik app sample file
  - Qlik Pluralsight Example - [External].qvf

- GraphQL API_Qlik Integration_Sample.qvf

FAQ

Q: TLDR. Got anything to quickly get started?
A: Qlik Pluralsight Example - [External].qvf. Download. Open. Replace API token in Queries. Profit.

Q: Does the GraphQL API documentation cover data on SKILLS and FLOW plans?
A: Not at this time. It is only available for SKILLS data, though a plan that has both SKILLS and FLOW products can utilize the API to pull SKILLS data.

Q: Is there a limitation to the number of queries we create?
A: There is no limit to the number of queries. We do ask that you create enough queries to satisfy your needs, though to omit large data pulls through the use of filters or pagination. This reduces load on our servers and provides a better experience to all customers who are using the API endpoint.

Q: How do I request an API key?
A: This can only be done on the Developer Portal that is listed in the Resource section by a Plan Admin within the account. If you do not know who your Plan Admin(s) is/are, please reach out to your dedicated CSM or support resource to identify them.

Q: How do I request a multi-plan API key?
A: To request an API key to handle multiple plans, please reach out to us at professionalservices@pluralsight.com. As long as each plan has an Integrations or ProServ SKU, we will be able to add it to a master API token for you.

Q: How long does it take to generate an API token?
A: After submitting a request through the Developer Portal link in the resource section, your API will be approved and ready to use immediately. If there is a delay for any reason or it does not function, please contact professionalservices@pluralsight.com.

Q: Will you create and maintain the Qlik Dashboards for our organization?
A: Not at this time. We may support a templated dashboard sample in the future.

Q: Does this cost extra?
A: If you have purchased the Integration's SKU, or have an Enterprise SKU that was purchased before July 1st 2020 and it has not yet expired and been renewed, then there is no additional charge. If you have not purchased our Integrations SKU and

have a subscription starting after July 1st 2020, whether a new subscription or renewal, then you will have to speak to your CSM and Sales rep to get started.

Q: What data can I not see?
A: If it is not listed in the Developer Portal documentation schema, then it is not available. Keep an eye on the change log section of the webpage for updates.

Q: I have resources to maintain the .qvf file, but I don't have resources to dedicate to development from scratch due to time-constraints/business priorities? Do you have any options to assist?
A: We do. Qlik Pluralsight Example - [External]..qvf file has multiple preloaded queries already set and ready to use for analytic use. All you would need to do is replace the API token that is present in each query in the file with your own token. It will then pull data for you and should remove the bulk of the needed query and data collection setup work. All you would need to do then is have a user create the reports that you are wanting with the data that is available.

Q: I do not like using GraphQL APIs or are unfamiliar with them. Can I continue to use your REST APIs?
A: You may continue to use our SKILLS REST APIs currently, however they will be turned off eventually. Our legacy REST APIs are being decommissioned sometime in Q4 2020/Q1 2021. The data availability and manipulation through the REST APIs is not as robust or long term supporting for our organization as we would have liked. Due to the limitations in REST, we have moved to development and support of GraphQL. We will be using GraphQL APIs as the standard for the SKILLS product and it will be the only option we support going forward. FLOW (formerly GitPrime) will continue to use REST. If you are using REST APIs currently for SKILLS data and want to know how to make the switch to GraphQL, please see the help documentation here or email us at professionalservices@pluralsight.com.