..

Data API

Date published: 2020-10-30
Date modified: 2022-09-21

https://docs.cloudera.com/

Legal Notice

Contents

Data API

CDP Data Visualization provides an interface for applications that fetch data through a REST interface.

Typically, users access CDP Data Visualization's web server through a web browser, and the interface is entirely visual. This visual interface interacts directly with CDP Data Visualization's data layer, which in turn provides seamless access to modeled data, integrated security controls, fast access to data in connected systems, and access to advanced analytics.

Sometimes, operational data applications do not need access to CDP Data Visualization's visual layer, but require direct access to the data layer. For these purposes, Arcadia supports a Data API that provides operational data applications with direct access to data.

See the following articles for more information about the Data API interface:

- Enabling data API
- Example of data API usage
- Data API request payload
- Data API response payload
- Accessing Data API Request Payload

Related Information

Example of data API usage

Data API request payload

Data API response payload

Data API request payload

Before invoking the Arcadia Data API, you must familiarize yourself with the syntax of the request payload.

Syntax of a Request Payload

The Data API has a consistent pattern for each payload, with the following basic access syntax:

```
{
        "version":version_number,
        "type":"SQL",
        "limit":number_of_rows,
        "dimensions":
          [{
             "type":"SIMPLE/SEGMENT",
             "expr":"[dimension_1] as 'dimension_value_1',
             "expr":"[dimension_2] as 'dimension_value_2',
             "expr":"[dimension_n] as 'dimension_value_n',
             "order": {"asc": True/False, "pri": order_sequence}
          }],
        "aggregates":
          [{
             "expr":["aggregate_1] as 'aggregate_value_1',
             "expr":["aggregate_2] as 'aggregate_value_2',
             "expr":["aggregate_n] as 'aggregate_value_n',
          }],
        "filters":
           [
             "[filter_1] in ('filter_value_1')",
             "[filter_2] in ('filter_value_2')",
```

```
              "[filter_n] in ('filter_value_n')"
           ],
        "having":
           [
            "[aggregate_1]) > aggregate_value_1"
           ],
        "dataset_id":dataset_id_number
}
```

Parameters of a Request Payload

The request payload parameters are defined as follows:

version

Specifies the version of the API. This is used for backward compatibility in case the API changes in the future. This is a mandatory parameter. For example, version 1.

type

Specifies the type of query. This is a mandatory parameter. For example, SQL.

limit

Specifies the number of rows to return. This is a mandatory parameter.

dimensions

List of zero or more dimensions. Specifying no dimensions or aggregates returns all columns from the dataset. Each item in the list specifies a dimension that is requested, and has the following structure of key value pairs:

- *type*

  Either SIMPLE or SEGMENT. Default is SIMPLE. Use field descriptions for expr and order with SIMPLE type.
- *expr*

  Either a dimension or an aggregate with the following format: "expr":"[dimension_1] as 'dimension_value'. This is a mandatory parameter.
- *order*

  Maximum of two key value pairs: ‘asc’, which is set to True or False. This indicates the order and priority, which is set to an integer and specifies the sequence in which the order is to be applied.

aggregates

List of zero or more aggregates. Specifying no aggregates returns all columns from the dataset. Each item in the list specifies an aggregate that is requested, and has the following structure of key value pairs:

- *expr*

  Specifies one aggregate with the following format:

  ```
  [{"expr":["aggregate_1] as 'aggregate_value_1'}]
  ```

dataset_id

Specifies the ID of the dataset object to run the data query. This is a mandatory parameter.

filters

Specifies a comma-separated list of filter expressions that apply to the dataset using the ‘WHERE’ clause. All the filters specified are combined with an ‘AND’ operation. These filters are applied as the ‘WHERE clause in the generated SQL. This is an optional parameter.

having

Specifies expressions to filter the dataset using the HAVING clause. Therefore, this parameter has an aggregation comparison. This is an optional parameter.

Example of a Request Payload

Here is an example of a Data API request payload that interfaces with the Cereals dataset, which ships as a sample with most Arcadia Enterprise installations.

```
{
        "version":1,
        "type":"SQL",
        "limit":100,
        "dimensions":
          [{
             "type":"SIMPLE",
             "expr":"[manufacturer] as 'manufacturer'",
             "order": {"asc":False, "pri":1
          }],
        "aggregates":
          [{
             "expr":["avg([fat_grams]) as 'avg(fat_grams)'",
             "expr":["sum(1) as 'Record Count'"
          }],
        "filters":
           [
             ["[cold_or_hot] in ('C')"]
           ],
        "having":
           [
            ["[avg([fat_grams]) < 5"]
           ],
        "dataset_id":11
```

Data API response payload

After a successful data request payload execution, the system returns an HTTP response of type text/json.

Syntax of a Response Payload

The Data API has a consistent pattern for an output payload, with the following basic access syntax:

```
{
  "info": [
    "Query:SELECT query
  ],
  "coltypes": [
    "type_of_single_column",
    "BIGINT"
  ],

  "colnames":[
    "column_name_1",
    "column_name_2",
    "column_name_n"
  ],

  "rows": [
    [
```

```
      "row_information_1",
      "row_information_2",
      "row_information_n"
    ]
  ]
}
```

Parameters of a Response Payload

The response payload parameters are defined as follows:

info

String of the raw SQL query that is executed.

coltypes

Array of strings, where each string specifies the type of a single column returned by the data request.

colnames

Array of strings, where each string specifies the name of a single column returned by the data request. This contains the alias for the columns as specified in the initial request.

rows

Array of arrays, where each inner array represents a single row of information returned by the data request.

Example of a Response Payload

Here is an example of a Data API response payload:

```
{
  "info": [
    "Query:SELECT TA_0.`manufacturer` as `manufacturer`, sum(1) as `Record
 Count`\n
       FROM `main`.`cereals`  TA_0\n  WHERE TA_0.`cold_or_hot` in ('C')\n  G
ROUP BY 1\n  LIMIT 100"
  ],
  "coltypes": [
    "STRING",
    "BIGINT"
  ],
  "colnames": [
    "manufacturer",
    "Record Count"
  ],
  "rows": [
    [
      "General Mills",
      22
    ],
    [
      "Kelloggs",
      23
    ]
  ]
}
```

Example of data API usage

To use the Data API interface in , you must enable the Data API, obtain an API key, and then invoke the Data API.

The API provides no discovery interfaces to understand the structure and format of the data. The invoker of the interface must be familiar with the target dataset, and what dimensions, aggregates, and filters are appropriate for each use case.

To construct the request payload, see Data API request payload.

The following example of a Data API python code interfaces with the Cereals dataset that ships as a sample within most Arcadia Enterprise installations.

Data API Programmatic Access

In this example, we are supplying the API Key that authorizes the user to access the dataset.

The host and port variables specify the running instance of the DataViz, and the api_key string has been obtained earlier, as described in Enabling Data API.

Note that you must replace the sample data request, dsreq, with your custom request. The dataset ID in this example is 11; it may be different on your system. Therefore, edit the last line of the dsreq specification to use the dataset ID of your target dataset.

```
import requests
import json
url = 'http://host:port/arc/api/data'

def _fetch_data(dsreq):
  headers = {
   'Authorization': 'apikey api_key'
  }

  params = {
    'version': 1,
    'dsreq': dsreq,
  }

  r = requests.post(url, headers=headers, data=params)
  if r.status_code != 200:
    print 'Error', r.status_code, r.content
    return
  raw = r.content
  d = json.loads(raw)
  print '\nData Request being sent is:\n', \
        json.dumps(json.loads(dsreq), indent=2)
  print '\nData Response returned is:\n', json.dumps(d, indent=2)

def main():

 # CHANGE the following dsreq to a data request of your choice.

  dsreq =
"""{"version":1,"type":"SQL","limit":100,
"dimensions":[{"type":"SIMPLE","expr":"[manufacturer] as 'manufacturer'"}],
"aggregates":[{"expr":"sum([sodium_mg]) as 'sum(sodium_mg)'"},
{"expr":"avg([fat_grams]) as 'avg(fat_grams)'"},
{"expr":"sum(1) as 'Record Count'"}],
"filters":["[cold_or_hot] in ('C')"],
"dataset_id":11}"""

  _fetch_data(dsreq)
```

```
if __name__ == '__main__':
  main()
```

Related Information

Data API request payload