Sources

A **source** is the raw data that you want to use to create a predictive model. A **source** is usually a (big) file in a comma separated values (CSV) format. See the example below. Each row represents an instance (or example). Each column in the file represents a **feature** or **field**. The last column usually represents the **class** or **objective field**. The file might have a first row named **header** with a name for each **field**.

```bash
Plan,Talk,Text,Purchases,Data,Age,Churn?
family,148,72,0,33.6,50,TRUE
business,85,66,0,26.6,31,FALSE
business,83,64,0,23.3,32,TRUE
individual,9,66,94,28.1,21,FALSE
family,15,0,0,35.3,29,FALSE
individual,66,72,175,25.8,51,TRUE
business,0,0,0,30,32,TRUE
family,18,84,230,45.8,31,TRUE
individual,71,110,240,45.4,54,TRUE
family,59,64,0,27.4,40,FALSE
```

A **source**:

1. Should be a comma-separated values (CSV) file. Spaces, tabs, semicolons and tabs are also valid separators.
2. Weka's ARFF files are also supported.
3. JSON in a few formats is also supported. See below for more details.
4. Microsoft Excel files or Mac OS numbers files should also work most times. But it would be better if you please export them to CSV (commad-separated values).
5. BigML will recognize images, assigning to them the "image" format, and extract from them a single row with two columns, namely, the image data (with type image), and its filename (with type path). All the magic needed to extract information from an image is already built-in in the platform, so you will not need to attach to an image source any parsing parameter. Once created, they are basically inmutable.

   Although you can use an image source to create a dataset with those two columns and a single row, that will not be in general very useful. Most often, you will want to have a collection of images, and create a dataset where every row represents one of them. See BigML's composite sources.
6. Cannot be bigger than **64GB**.

7. Compressed files (zip, gzip, bz2, tar and tar.gz) containing more than one file are also supported. BigML will then automatically create one source for each file inside the archive, and put them all together in a composite source.

You can also create sources from remote locations using a variety of protocols like **https**, **hdfs**, **s3**, **asv**, **odata/odatas**, **dropbox**, **gcs**, **gdrive**, etc. See below for more details.

BigML.io allows you to create, retrieve, update, and delete your **source**. You can also list all of your **sources**.

Jump to:

JSON Sources

BigML.io can parse JSON data in one of the following formats:

1. A top-level list of lists of atomic values, each one defining a row

```json
[
  ["length","width","height","weight","type"],
  [5.1,3.5,1.4,0.2,"A"],
  [4.9,3.0,1.4,0.2,"B"],
  ...
]
```

2. A top-level list of dictionaries, where each dictionary's values represent the row values and the corresponding keys the column names. The first dictionary defines the keys that will be selected.

```json
[
  {"length":5.1,"width":3.5,"height":1.4,"weight":0.2,"type":"A"},
  {"length":4.9,"width":3.0,"height":1.4,"weight":0.2,"type":"B"},
  ...
]
```

3. A top-level list of dictionaries with the request parameter json_key defined under **source_parser** with the value of one of its keys having any of the two formats above. For the following example, you can set "source_parser": {"json_key": "data"}.

```json
{
  "name": "Shipping Class",
  "data": [
      {"length":5.1,"width":3.5,"height":1.4,"weight":0.2,"type":"A"},
      {"length":4.9,"width":3.0,"height":1.4,"weight":0.2,"type":"B"},
      ...
  ]
}
```

Alternatively, if the source is from a remote location and is going to be downloaded from, say, http://yourcompany.com?a=foo&b=bar and the data rows will come under the key "data" like the example above, you could request an external source with **bigml_json_key** in the URL: "http://yourcompany.com?a=foo&b=bar&bigml_json_key=data". Note that the parameter name in the query string is **bigm_json_key**, not **json_key**.

4. A nested dictionary key, with the final value having any of the formats already described. For the following example, you can set "source_parser": {"json_key": "results.data"}.

```json
{
  "name": "Shipping Class",
  "results": {
      "meta": "Shipping class info",
      "data": [
          {"length":5.1,"width":3.5,"height":1.4,"weight":0.2,"type":"A"},
          {"length":4.9,"width":3.0,"height":1.4,"weight":0.2,"type":"B"},
          ...
      ]
  }
}
```

Like the example with the top-level list of dictionaries, you could request an external source with **bigml_json_key** in the URL for a remote source. For the example above, you could request "http://yourcompany.com?a=foo&b=bar&bigml_json_key=results.data".

5. A top-level dictionary of dictionaries whose values represent rows

```json
{
  "GnCC": {"length":5.1,"width":3.5,"height":1.4,"weight":0.2,"type":"A"},
  "4R3R": {"length":4.9,"width":3.0,"height":1.4,"weight":0.2,"type":"B"},
  ...
}
```

6. Rows of JSON dictionaries where the full file is not a valid JSON document, but each individual line is. Each line in the file (or input stream) must be a separate JSON either list or map per line, and is thus parsed individually as a top-level JSON document, and interpreted as a row of data.

To be a valid JSON row, each line must fall into one of two categories:

- **A JSON dictionary**, with at least some of its values being atomic. In that case, the keys in the dictionary are taken as the field names and the corresponding atomic values as the actual columns of the row. Keys with values that are composite are just ignored. The first map in the file determines what are the fields that will be extracted in subsequent rows unless they are specified via the request parameter json_fields defined under **source_parser** or the query string parameter **bigml_json_fields** just as with the **json_key**

explained above.

- **A JSON list**, again with at list some of its values being atomic. Here, the field names can be inferred from the values of the first list in the file, using the same heuristics that are used to auto-detect headers in CSVs. Or you can set the **header** flag as well as use **json_fields** under **source_parser** (or **bigml_json_fields** in the query string) to give explicit names in the creation request. Non-atomic values appearing in the lists are translated to missing values.

Here's a snippet of JSON rows using maps:

```json
{"length":5.1,"width":3.5,"height":1.4,"weight":0.2,"type":"A"}
{"length":4.9,"width":3.0,"height":1.4,"weight":0.2,"type":"B"}
{"length":4.7,"width":3.2,"height":1.3,"weight":0.2,"type":"B"}
{"length":4.6,"width":3.1,"height":1.5,"weight":0.2,"type":"C"}
{"length":5.0,"width":3.6,"height":1.4,"weight":0.2,"type":"A"}
{"length":5.4,"width":3.9,"height":1.7,"weight":0.4,"type":"C"}
...
```

and here's JSON rows with lists:

```json
["length","width","height","weight","type"]
[5.1,3.5,1.4,0.2,"A"]
[4.9,3.0,1.4,0.2,"B"]
[4.7,3.2,1.3,0.2,"B"]
[4.6,3.1,1.5,0.2,"C"]
[5.0,3.6,1.4,0.2,"A"]
[5.4,3.9,1.7,0.4,"C"]
...
```

Open vs. closed sources

Once the raw source data is in BigML's servers, a source resource is created, which specifies how to extract rows from it to create a dataset. BigML tries to automatically figure out how to do that. For instance, if your file is a CSV file (maybe zipped), or an ARFF file or a JSON file with an adequate format, it will most of the time discover what columns and column types it has. This information is exposed in the source's fields.

If BigML doesn't get it right, you can edit that metadata and adjust the definitions of the source's fields, until they are correct. That is why there is a separate source resource instead of just a dataset: it represents the

configuration needed to create datasets and it's the only resource in the platform that is initially mutable.

The source is open when it can be modified. All sources are initially created open. Once you are happy with their definition, you make them immutable by either explicitly marking them as closed or (if they are not composites, see below) creating a dataset with them, which closes them for you.

```curl
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{"closed": true}'
```

Once a source has been closed, it cannot be modified: that way, you can always know how you created your datasets and retrace the full chain of steps leading to any resource in the platform (all other BigML resources are immutable: they are always closed).

What if you discover that you want to change something in your original source? You can clone it very easily, creating a new source (**which will be open and modifiable**) resource that shares the original underlying data, BigML doesn't need to fetch it again nor do you need to upload it again! Cloning is therefore very quick and efficient. Then you can change the parsing specifications in your cloned source, and go on to dataset creation and beyond when it's ready.

It's also possible to clone an open source.

```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -X POST \
    -H 'content-type: application/json' \
    -d '{"origin": 603e1fef1f386f43db000000"}'
```

Composite sources

A composite source is an arbitrary collection of other sources, of any kind. When all the sources inside the composite have the same fields, the composite itself will inherit those fields, and you will be able to create a dataset from it: the result will just be the concatenation of all the rows extracted from each component source inside the composite.

You could put together a list of CSV sources, or maybe a couple of CSV files and an ARFF file with the same exact fields, and the resulting composite will inherit those fields and behave like a single source for all practical purposes.

As any other source, a (possibly empty) composite is created open, meaning that you can modify it. In the case of composites, modifying it means performing one of the following operations:

The sources in a composite are called *components*.

components arguments accept both formats of a valid source ID, either **source/602521458c10327ecab82b9c** or **602521458c10327ecab82b9b**.

1. Adding components.

curl

```
curl "https://bigml.io/andromeda/source/6034c44a1f386fc650000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{"add_sources": [
            "source/602521458c10327ecab82b9c",
            "602521458c10327ecab82b9b"
        ]}'
```

2. Removing components.

curl

```
curl "https://bigml.io/andromeda/source/6034c44a1f386fc650000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{"remove_sources": [
            "source/602521458c10327ecab82b9c",
            "602521458c10327ecab82b9b"
        ]}'
```

3. Replacing the full list of components.

curl

```
curl "https://bigml.io/andromeda/source/6034c44a1f386fc650000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{"sources": [
            "source/602521458c10327ecab82b9c",
            "602521458c10327ecab82b9b"
        ]}'
```

A source can belong to as many composites as you wish, and composites can be nested, with the only limitation that a composite can only be a component if it's closed.

When a source belongs to one or more composites, it cannot be modified, regardless of whether it's open or closed. That way all composites see the same version of the source all the time.

As you add or remove components to a composite, it will check the compatibility of the fields of all its components, and update its own set of fields. Thus, adding and removing sources to a composite is in this sense analogous to changing the parsing specification of, say, a CSV, in the sense that that is also an operation that can potentially change the collection of fields (and even the number of rows) extracted to the CSV.

Once you have finished adding components to a composite and want to use it to create datasets, you must close it. When you close a composite, all its components will be automatically closed for you.

Unlike all other kinds of source, composites created this way must be explicitly closed by an API call or UI action in order to create a dataset. That is mainly to avoid accidentally closing a composite that is being worked on by several collaborators, or by mistake. Since composites can have a huge number of components and closing them also closes all of them, it may be relatively slow.

Creating composites from single zip files

As an alternative to combining pre-existing sources into a composite, one can also upload a zip or tar file containing more than one file. BigML will then automatically create one source for each file inside the archive, and put them all together in a composite source.

Image composites, extracted features and labels

If you put together a bunch of image sources inside a composite, that composite will also have format "image", and if you create a dataset from it, every row will correspond to one of the images in the composite, and have a column representing the image data, and another its filename.

Extracted features

But that's not all. BigML knows how to analyze the image in each row and automatically generate a set of numeric features for each one. Those features will appear as new auto generated fields in the composite. This process is analogous, for instance, to the way BigML will parse for you datetime values such as "2020-14-09" and generate a year, month and day additional fields in the final dataset.

By default, BigML will extract around two hundred features per image, representing its gradients histogram, and you can choose several others. These are all numeric values describing the structure and contents of

your image. By exposing them as explicit fields in your datasets, they can be used by all ML algorithms and models in the platform, just as any other feature, and you are not limited to their use in deep learning (which of course is also available, via our deepnets, which know how to deal directly with the raw image cells if you need it).

Label fields

Another common need when one has collections of images is to label them individually. In BigML, the natural way of representing those labels is as additional fields attached to the composite. This provides the additional flexibility of being able to have more than one label per image and having at your disposal multiple label types (categorical, numeric, items, text...).

You can thus add new label fields to any open composite of images, and then edit their values for each of the components inside. BigML's web UI provides an interface for doing it manually, and there are flexible ways in the API for updating them programmatically.

curl

```
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{
          "new_fields": [
            {
              "name": "Label",
              "optype": "numeric"
            }
          ]
        }'
```

curl

```
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{
          "row_values": [
            {
              "components": [
                "602521458c10327ecab82b9b",
                "602521458c10327ecab82b9c",
                "602521458c10327ecab82b9d"
              ],
              "field": "100002",
```

```
            "value": 6
        }
```

Label fields can only be added and edited to composites of images that are open.

Labeling using datasets

An alternative to using label fields in an image composite and editing their values at the source level is to create a dataset with just your images, another one with a CSV (or similar format) whose columns contain their labels, and then join the two by either image id or path name.

Since BigML has a very flexible set of dataset generation capabilities, including a full-fledged SQL engine, this is probably the best way of handling image labeling in the general case.

Besides the simple 'one labelled image per row' scenario, working at the dataset level will also cover cases where you want to have more than one image cell per row, or images that appear in more than one row, or combinations thereof. For simpler labeling cases, when a CSV file with their final data is available, one can alternatively work at the datasource level using the table+image format.

The table+image format

There is a very common use case in which one has a collection of images and a CSV file representing their labels, or rows which contain several images, potentially with other associated features.

In this scenario, what we have is, essentially, a table of values with some of their cells containing images, and what we want to create is a dataset from that CSV.

BigML allows the creation of such a 'CSV with images in their cells' source from a zip archive containing both a CSV file and all the images that appear in the columns of that CSV. They will be referenced there by their filenames or paths.

When the system opens a zip file, it will create the corresponding composite with all the sources inside. In this case, one CSV source, and a collection of image sources. It will then look for fields in the CSV that reference the images, and mark its format as "table+image". Note that this is a special case: since the components of the composite have different fields (one of them has the fields in the CSV, and the others only the image fields), the resulting combination shouldn't have by default any fields, and its format would be "mixed". Instead of that, BigML recognizes that this is a CSV with images, marks it as such and sets the fields of the composite to those of the CSV. It also attaches to each row auto-generated fields with the features extracted from its images.

So, by way of example, imagine you have a CSV with rows that look like:

```
cat_feature,file_1,file_2,num_feature,row_class
a,images/digits/3/7.png,images/digits/3/44.png,0.25891675029296335,digit_3
```

```
b,images/digits/3/86.png,images/digits/3/7.png,0.9677999949201714,digit_3
c,images/digits/3/130.png,images/digits/3/111.png,0.13927370519815263,digit_3
...
```

and you put it in a zip file together with image files inside the folder images and its subfolders. The platform will then create a source for each of the image files, and a source for the CSV, and put them together in a composite with format table+image whose fields will be cat_feature, image_1, image_2, num_feature, row_class, and a collection of auto-generated fields coming from image_1 and image_2 consisting of the image data in image_1.image, image_2.image and all the corresponding extracted features.

Then, when you create a dataset from this composite, BigML will use the CSV, and look at the values of its image cells to extract the corresponding values of its autogenerated features.

This is essentially what you would do if you uploaded the CSV on the one hand and created a dataset with it, the images on the other and created a second dataset will all of them, and then performed a join by the columns image_1 and image_2 in the CSV and filename in the images datasets, and finally added to the result auto-generated features (this can also be done at the dataset level using flatline).

It is important to notice that **composites with format table+image and composites with format image are quite different**. The latter represents a list of rows with an image per row, all of them different. The table+image, by contrast, represents a CSV file some of whose cells contain images. But those images can appear in more than one row, or there can be several in a single row. It is because of this different nature that **it is not possible to add new fields to a table+image composite, nor add or remove components (they're created closed, as immutable entities) from it: they can be created only from a zip file upload (or fetched from a remote location)**, just like any other CSV source.

Source Base URL

You can use the following base URL to create, retrieve, update, and delete **sources**.

```bash
https://bigml.io/andromeda/source
```

All requests to manage your **sources** must use HTTPS and be authenticated using your **username** and **API key** to verify your identity. See this section for more details.

Creating a Source

You can create a new source in any of the following five ways:

1. **Local Sources**: Using a local file. You need to post the file content in "multipart/form-data". The maximum size allowed is 64 GB per file.
2. **Remote Sources**: Using a URL that points to your data. The maximum size allowed is 64 GB or 5 TB if you use a file stored in Amazon S3.
3. **Inline Sources**: Using some inline data. The content type must be "application/json". The maximum size in this case is limited 10 MB per post.
4. **Synthetic Sources**: Automatically generate synthetic data sources, presumably for activities such as testing, prototyping, and benchmarking.
5. **External Data Sources**: Using connection data to an external data database or document repository.

Creating a Source Using a Local File

To create a new source, you need to POST the file containing your data to the **source** base URL. The file must be attached in the post as a file upload.The Content-Type in your HTTP request must be "multipart/form-data" according to RFC2388. This allows you to upload binary files in compressed format (.Z, .gz, etc) that will be uploaded faster.

You can easily do this using **curl**. The option -F (--form) lets curl emulate a filled-in form in which a user has pressed the submit button. You need to prefix the file path name with "@".

```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" -F file=@iris.csv
```

Creating a Source Using a URL

To create a new remote source you need a URL that points to the data file that you want BigML to download for you.

You can easily do this using **curl**. The option -H lets curl set the content type header while the option -X sets the http method. You can send the URL within a JSON object as follows:

```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -X POST \
    -H 'content-type: application/json' \
    -d '{"remote": "https://static.bigml.com/csv/iris.csv"}'
```

You can use the following types of URLs to create remote sources:

1. HTTP or HTTPS. They can also include basic realm authorization.

```
https://test:test@static.bigml.com/csv/iris.csv
http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
```

2. Public or private files in Amazon S3.

```
s3://bigml-public/csv/iris.csv
s3://bigml-test/csv/iris.csv?access-key=AKIAIF6IUYDYUQ7BALJQ&secret-key=XgrQV/hHBVymD75AhFO
```

Creating a remote source from Google Drive and Google Storage

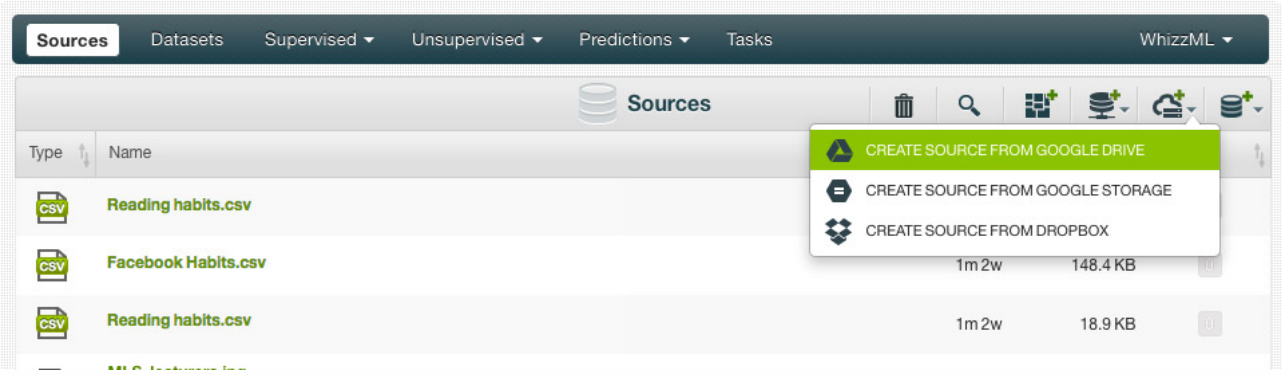
You have two options to create a remote datasource from Google Drive and Google Storage via API:

- Using BigML:

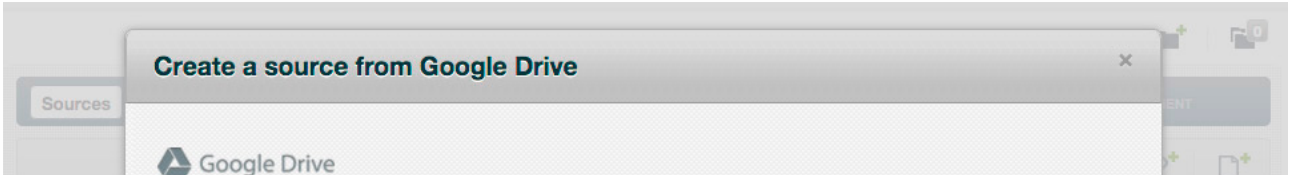
  Allow BigML to access to your **Google Drive** or **Google Storage** from the Cloud Storages section from your Account or from your Dashboard sources list. You will get the **access token** and the **refresh token**.

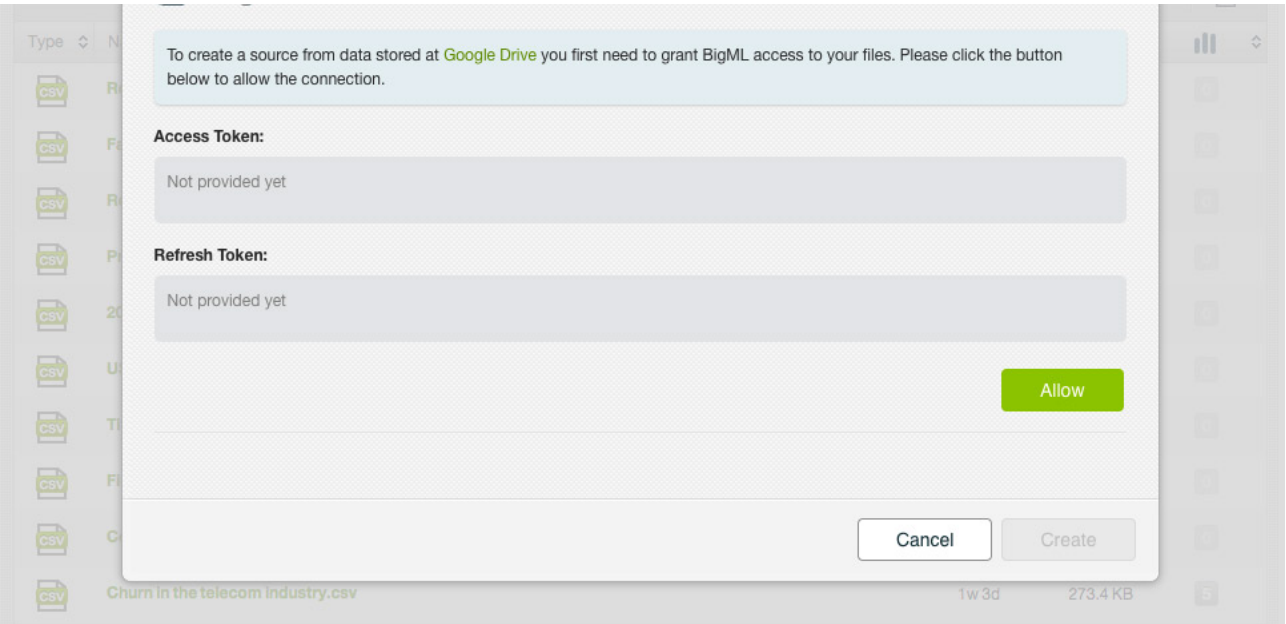
  Google Drive example:

  1. Select the option to create source from Google Drive:

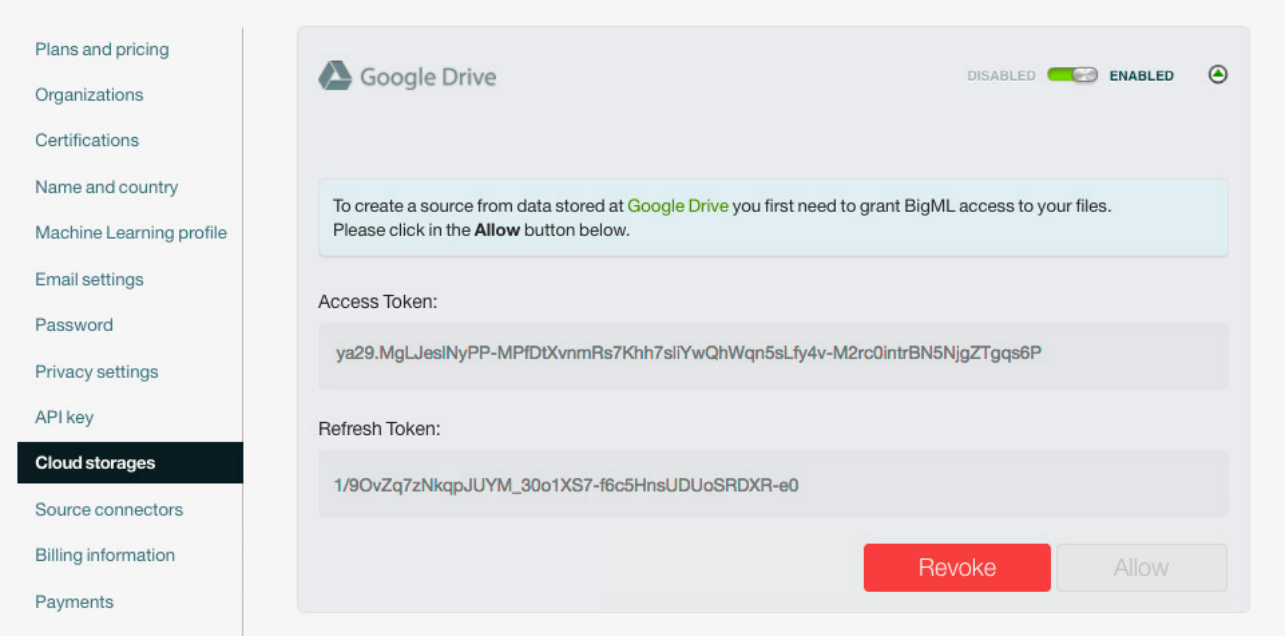


  2. Allow BigML access to your Google Drive:





3. Get the access token and refresh token:



After complete these steps you need to POST to the **source** endpoint URL an object containing at least the **file ID** (for Google Drive) or the **bucket** and the **file name** (for Google Storage) and the **access token**. Including also the **refresh token** is optional before your access **token** expires. Including it avoids you to be worried about expiration time. The content-type must always be **"application/json"**.

You can easily create the remote source using **curl** as in the examples below:

```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
     -X POST \
     -H 'content-type: application/json' \
     -d '{"remote":"gdrive://noserver/0BxGbAMhJezOScTFBUVFPMy1xT1E?token=ya29.AQGn2MO578
```

You should complete steps above in the same way to import source from Google Cloud Storage.

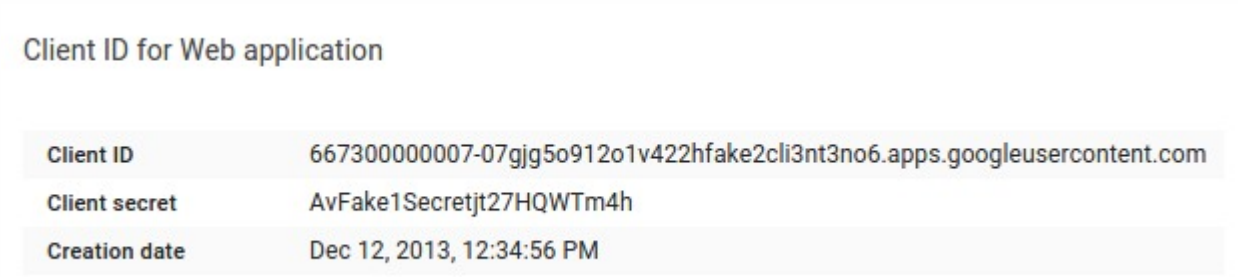
```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
     -X POST \
     -H 'content-type: application/json' \
     -d '{"remote":"gcs://company_bucket/Iris.csv?token=ya29.AQGn2MO578Fso0fVF0hGb0Q60IL(
```

- Using your own app:

You can also create a remote source from your own App. You first need to authorize BigML access from your own Google Apps application. BigML only needs authorization for read-only authentication scope (https://www.googleapis.com/auth/devstorage.read_only, https://www.googleapis.com/auth/drive.readonly), but you can have any of the other available scopes (find authentication scopes available for Google Drive and Google Storage). After the authorization process you will get your **access token** and **refresh token** from the Google Authorization Server.

Then the process is the same as creating a remote source using BigML application described above. You need to POST to the **source** endpoint an object containing at least the **file ID** (for Google Drive) or the **bucket** and the **file name** (for Google Storage) and the **access token**, but in this case you will also need to include the **app secret** and **app client** from your App. Again, including the refresh token is optional.

Your values for app client and app secret appear as **Client secret** and **Client ID** in Google developers console respectively. See image below.




```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
     -X POST \
     -H 'content-type: application/json' \
     -d '{"remote":"gdrive://noserver/0BxGbAMhJezOSXy1oRU5MSU90SUU?token=ya29.AQGn2MO578
```

```curl
```

```
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
     -X POST \
     -H 'content-type: application/json' \
     -d '{"remote":"gcs://company_bucket/Iris.csv?token=ya29.AQGn2MO578Fso0fVF0hGb0Q60IL(
```

Creating a Source Using Inline Data

You can also create sources sending some inline data within the body of a POST http request. This way is specially useful if you want to model small amounts of data generated by an application.

To create an inline source using **curl** you can use the following example:

curl

```
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -X POST \
    -H 'content-type: application/json' \
    -d '{"data": "a,b,c,d\n1,2,3,4\n5,6,7,8"}'
```

The **data** attribute expects its associated value to be a string. The example shows a string that includes a headers row plus two more rows. The new line characters between the rows act as line separators. Similarly, stringified JSON is accepted too:

curl

```
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -X POST \
    -H 'content-type: application/json' \
    -d '{"data": "[{\"length\":5.1,\"type\":\"A\"},{\"length\":4.9,\"type\":\"B\"}]"}'
```

Creating a Source with Automatically Generated Synthetic Data

You can also synthetically create sources using automatically generated data for activities such as testing, prototyping, and benchmarking.

To create a syntheric source using **curl** you can use the following example:

```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -X POST \
    -H 'content-type: application/json' \
    -d '{"synthetic": {"fields": 10, "rows": 10}}'
```

Creating a Source Using External Data

You can also create sources getting data from external data sources like databases or document repositories. You will need to pass connection data to those repositories with permissions to access the data to import in BigML as well as querying information to filter data to include in the source.

Argument	Type	Description
connection optional	Object	The connection info to data repository. Each repository manages its own parameters, see next subsections for details. **Example**: { "connection": { "master": "local", "database": "default" }, }
externalconnector_id optional	String	The id of the **external connector** object you want to use to import data from. **Example**: "602c199f1f386f44810001ff"
fields optional	String / Array	A list of fields or a string containing a comma-separated list of fields to include in the source. **Example**: ["field11","field2"]
fields_exclude optional	String / Array	A list of fields or a string containing a comma-separated list of fields **not** to include in the source. **Example**: ["field11","field2"]
limit optional	Integer, default is **10**	The number of records to use in the source. **Example**: 50

Argument	Type	Description
offset optional	Integer, default is **0**	The starting offset of records to use in the source. **Example**: 1
query optional	String / Object	Search definition using specific query syntax in data repository. • All **RDBMS**: SQL query. **Mandatory**. • **Elasticsearch**: Query in the Lucene query string syntax or search definition using the Query DSL. **Example**: "SELECT * FROM table_name"
sort optional	String / Array	A list of fields or a string containing a comma-separated list of [-]fields with sorting conditions. **Example**: ["field11","field2"]
source optional	String	Available data repositories: **mysql**, **postgresql**, **sqlserver** and **elasticsearch** **Example**: "elasticsearch"
tables optional	String / Array	A list of index/tables names to search, or a string containing a comma-separated list of index/tables names to search. **Example**: "table_name1,table_name2"

Note that for RDBMS connectors, the 'table', 'offset', 'sort', 'fields', and 'fields_exclude' parameters are ignored if a 'query' is provided. With respect to the 'limit', any limit within the provided 'query' will by applied first, before that of the 'limit' parameter value.

curl

```
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -X POST \
    -H 'content-type: application/json' \
    -d '{"external_data": {
            "source": "mysql",
            "connection": {
                "host":"mysql.host.com",
                "port": 3306,
                "database": "dbname",
                "user": "dbuser",
                "password": "dbpwd"
```

```
            },
            "query": "SELECT * FROM table_name"
        }}'
```

PostgreSQL, MySQL, SQL Server

All RDBMS connectors share connection parameters.

Argument	Type	Description
database	String	Name of database to initialize connection with. **Example**: "dbname"
host	String	Database host server. **Example**: "mysql.host.com"
password	String	Database user pssword. **Example**: "dbpwd"
port	Integer	TCP port for connecting to the database server. **Example**: 3306
user	String	Database user. **Example**: "dbuser"

Elasticsearch

Elasticsearch exposes several parameters to specify the connection to a server, including the hosts, authenticaton info, ssl config, etc. See Elasticsearch for complete reference.

Argument	Type	Description
ca_certs optional	String	Optional path to CA bundle. See https://urllib3.readthedocs.io/en/latest/security.html#using-certifi-with-urllib3 for instructions how to get default set.
client_cert optional	String	Path to the file containing the private key and the certificate, or cert only if using client_key.

Argument	Type	Description
client_key optional	String	Path to the file containing the private key if using separate cert and key files (client_cert will contain only the cert).
headers optional	Object	Any custom http headers to be add to requests.
hosts	String / Array	A list of nodes to connect to. Node should be a dictionary ({"host": "localhost", "port": 9200}) or a string in the format of host[:port]. **Example**: [{“host”: “localhost”, “port”: 9200}, "localhost:9200"]
http_compress optional	Boolean	Use gzip compression.
maxsize optional	Integer	The number of connections which will be kept open to this host. See https://urllib3.readthedocs.io/en/1.4/pools.html#api for more information.
port optional	Integer, default is **9200**	Port to use. **Example**: 443
ssl_assert_fingerprint optional	String	Verify the supplied certificate fingerprint if not None.
ssl_assert_hostname optional	Boolean	Use hostname verification if not False.
ssl_show_warn optional	Boolean	Show warning when verify certs is disabled.
ssl_version optional	String	version of the SSL protocol to use. Choices are: SSLv23 (default) SSLv2 SSLv3 TLSv1 (see PROTOCOL_* constants in the ssl module for exact options for your environment).
timeout optional	Number, default is **10**	Default timeout in seconds.

Argument	Type	Description
url_prefix optional	String	Url prefix for elasticsearch.
use_ssl optional	Boolean	Use ssl for the connection.
verify_certs optional	Boolean	Whether to verify SSL certificates.

Independently of how you create a new source (local, remote, inline, synthetic or external data) **BigML.io** will return a newly created **source** document, if the request succeeded.

```json
{
    "category": 0,
    "closed": null,
    "code": 201,
    "configuration": null,
    "configuration_status": false,
    "content_type": "unknown",
    "created": "2021-03-02T11:22:23.230728",
    "creator": "alfred",
    "description": "",
    "disable_autolabel": false,
    "disable_datetime": false,
    "field_types": {
        "auto_generated": {},
```

Source Arguments

In addition to the file, you can also use the following arguments.

Argument	Type	Description
add_sources optional	Array of Strings	A list of source identifiers to add to this composite. **Example**: ["602521458c10327ecab82b9c", "source/602521458c10327ecab82b9b"]

Argument	Type	Description
category optional	Integer, default is **0**	The category that best describes the **source**. See the category codes for the complete list of categories. **Example**: 1
closed optional	Boolean, default is **false**	Whether to close an open source (note that closed sources cannot be re-opened, they must be cloned). **Example**: true
data optional	String	Data for inline source creation. **Example**: "a,b,c,d\n1,2,3,4\n5,6,7,8"
delete_sources optional	Array of Strings	A list of source identifiers to remove from the composite and, if they do not belong to any other composite, delete as individual resources. **Example**: ["602521458c10327ecab82b9c", "source/602521458c10327ecab82b9b"]
description optional	String	A description of the **source** up to 8192 characters long. **Example**: "This is a description of my new source"
disable_datetime optional	Boolean, default is **false**	Whether BigML has to generate or not new fields from existing date-time fields. **Example**: true
external_data optional	Object	Set of parameters to generate a **source** from external data. See this section.
file optional	multipart/form-data; charset=utf-8	File containing your data in csv format. It can be compressed, gzipped, or zipped if the archive contains only one file
image_analysis optional	Object, default is shown in the table below	Configuration for analysis of image data in the **source**. **Example**: { "enabled": "true", "extracted_features": ["average_pixels"], "maximum_dimension": 32 }
		Configuration for analysis of image data in the **source**. **Example**:

Argument	Type	Description
item_analysis optional	Object, default is shown in the table below	{ "separator": ";", "limit": 100, "pruning_strategy": "most_frequent", "target_frequency": 0.5 }
name optional	String default is **Unnamed source**	The name you want to give to the new **source**. **Example**: "my new source"
new_fields optional	Array	List of label fields to add to image composites, with their name and optype. Names must be unique and not used by any other field, and all optypes except image are allowed. See the table below for more details. **Example**: [{ "name": "color", "optype": "categorical" }]
origin optional	String	The **source/id** of the source to be cloned. **Example**: "source/603e1fef1f386f43db000000"
project optional	String	The **project/id** you want the **source** to belong to. **Example**: "project/603de73d1f386f7360000000"
remote optional	String	A URL pointing to file containing your data in csv format. It can be compressed, gzipped, or zipped. **Example**: "https://static.bigml.com/csv/iris.csv"
remove_fields optional	Array	List of fields to remove from existing label fields (previously added via new_fields). **Example**: ["color"]
remove_sources optional	Array of Strings	A list of source identifiers to remove from this composite. **Example**: ["602521458c10327ecab82b9c", "source/602521458c10327ecab82b9b"]
		Default list of source identifiers for row_values entries without

Argument	Type	Description
row_components optional	Array of IDs	components or indices of their own. "all" is also an allowed value. **Example**: ["602521458c10327ecab82b9c", "source/602521458c10327ecab82b9b"]
row_indices optional	Array of IDs	Default list of indices for row_values entries without components or indices of their own. Ranges and "all" are also allowed values. **Example**: ["602521458c10327ecab82b9c", "source/602521458c10327ecab82b9b"]
row_values optional	Array of Objects	A list of maps specifying values for different rows and label fields via the following properties: • **indices**: a list of index values as single integers or as a pair of the form [from, to] denoting a closed interval. This property can also take the string value "all", to denote all rows. • **components**: a list of source identifiers, denoting also components to which the update applies. • **value**: is the value to be assigned to the given field and rows. • **field**: the field identifier or field name to which the given value is to be assigned. The rows in which the given field gets the given value will be the union of the components and indices specs.
source_parser optional	Object, default is **shown in the table below**	Set of parameters to parse the **source**. **Example**: { "header": true, "locale": "en-US", "missing_tokens": ["?"], "quote": """, "separator": ",", "trim": true }
sources optional	Array of Strings	New list of source identifiers in this composite source, which will replace the previous one. **Example**: ["602521458c10327ecab82b9c",

Argument	Type	Description
		"source/602521458c10327ecab82b9b"]
synthetic optional	Object, default is shown in the table below	Set of parameters to generate a synthetic **source**. **Example**: { "fields": 1', "fields": 10, "missing": 0.1 }
tags optional	Array of Strings	A list of strings that help classify and index your **source**. **Example**: ["best customers", "2021"]
term_analysis optional	Object, default is shown in the table below	Set of parameters to activate text analysis for the **source**. **Example**: { "enabled": true, "ngrams": 5, "stem_words": true, "case_sensitive": false, "language": "en", "token_mode": "tokens_only" }
transformations optional	Array	A list of transformations to be performed on the original **source**. See Image transformations section. **Example**: [{ "operation": "crop", "arguments": { "x1": 0, "y1": 10, "x2": 100, "y2": 200 }, ... }]
webhook optional	Object	A webhook url and an optional secret phrase. See the Section on Webhooks for more details. **Example**: { "url": "http://myhost/path/to/webhook",

Argument	Type	Description
		"secret": "mysecret" }

Note that at least "file", "remote" or "data" must be present.

Source Parser Object

A **source parser** object is composed of any combination of the following properties.

Property	Type	Description
header optional	Boolean, default is **true**	Whether the source contains a header or not. **Example**: true
json_fields	Array of Strings	The columns to be used when the source is in the JSON format and the rows are a list of dictionaries. See the JSON Sources for more information. **Example**: ["age", "height", "weight"]
json_key	String	A top-level dictionary key containing the rows when the source is the JSON format. See the JSON Sources for more information. **Example**: "data"
locale optional	String, default is **en-US**	The locale of the **source**. **Example**: "es-ES"
missing_tokens optional	Array of Strings, default is **["", "N/A", "n/a", "NULL","null", "-", "#DIV/0", "#REF!","#NAME?", "NIL", "nil", "NA", "na", "#VALUE!", "#NULL!","NaN", "#N/A", "#NUM!", "?"]**	Tokens that represent a missing value. **Example**: ["?"]
quote	Char,	The source quote character.

Property	Type	Description
optional	default is **"""**	**Example**: ""
separator optional	Char, default is **,**	The source separator character. Empty string if the source has only single column. **Example**: ";"
trim optional	Boolean, default is **true**	Whether to trim field strings or not. **Example**: true

You can also use **curl** to customize your new **source** with a name and different parser. For example, to create a new source named "my source", without a **header** and with "x" as the only missing token.

```curl
curl "https://bigml.io/andromeda/source?$BIGML_AUTH" \
    -F file=@iris.csv \
    -F 'name=my source' \
    -F 'source_parser={"header": false, "missing_tokens":["x"]}'
```

If you do not specify a name, **BigML.io** will assign to the **source** the same name as the file that you uploaded. If you do not specify a **source_parser**, **BigML.io** will do its best to automatically select the parsing parameters for you. However, if you do specify it, **BigML.io** will not try to second-guess you.

New Fields Object

Property	Type	Description
description optional	String	A description for the new field. **Example**: "This field is a label"
item_analysis optional	Object	Parameters to activate item analysis in this field, when it has optype items.See the Section on Item Analysis for more details. **Example**: { "separator": ";", "limit": 100, "pruning_strategy": "most_frequent", "target_frequency": 0.5 }

Property	Type	Description
label optional	String	Label of the new field. **Example**: "New label"
name optional	String	Name of the new field. **Example**: "label"
optype optional	String	Optype of the new field. Available optypes are "numeric", "categorical", "text", "datetime", "items", "image", "path", and "regions". **Example**: "text"
term_analysis optional	Object	Set of parameters to activate text analysis for the dataset. See the Section on Term Analysis for more details. **Example**: { "enabled": true, "ngrams": 5, "stem_words": true, "case_sensitive": false, "language": "en", "token_mode": "tokens_only" }

Image Analysis

An **image_analysis** object is composed of any combination of the following properties.

Property	Type	Description
enabled optional	Boolean, default is **false**	Whether image analysis should be performed. **Example**: true
		A list of feature set names, or feature set names followed by parameter values. Available feature sets are: • **dimensions**: Generates four fields, corresponding to the raw image size, width, height and aspect ratio. • average_pixels: Generates fields for the red, blue, and green pixel values for several extremely low resolution

Property	Type	Description
extracted_features optional	Array	versions of the image (1x1, 3x3 and 4x4). This is fast to calculate and captures the high-level spatial and color information, but all detail is lost. • level_histogram: Generates fields for the color information in each channel divided into 16 equally spaced histogram bins. Each color histogram is normalized so that all values are between 0 and 1. While this gives very detailed color intensity information, all spatial information is lost. • histogram_of_gradients: Computes a histogram of oriented gradients for the entire image, and for all subimages on a 3x3 and 4x4 grid. The histograms are normalized within each subimage, so that all values are between 0 and 1. This histogram generally captures useful spatial and detail information, but precise color information is lost. Generally, this extractor is good at classifiying different shapes, or images where the orientation of the edges is a defining characteristic. • pretrained_cnn: Uses the top layer before the softmax output of an image-net pretrained CNN as the input features. The CNN to be used is a parameter to the extractor. Currently available pretrained networks are mobilenet, mobilenetv2, resnet18, resnet50, and xception. Note that the number of features output depends on the network used. • wavelet_subbands: Performs an n level Haar wavelet decomposition on the image, where n is a parameter. This parameter determines the number of recursive compositions that the featurizer will undertake, and so determines the number of output features. After decomposition, the pixels in each subband are aggregated using mean and standard deviation for both the full images and a 2x2 grid. Since each subband contains all image detail at a certain resolution in one of three directions, this feature type contains both spatial and frequency domain information about the nature of the detail in the image, but the directionality of the detail is only coarsely captured (contrast histogram_of_gradients). Typically useful for problems where texture is a defining characteristic of the image, or where there is obvious periodicity **Example**: ["dimensions",

Property	Type	Description
		"average_pixels", "level_histogram", "histogram_of_gradients", ["pretrained_cnn", "mobilenetv2"], ["wavelet_subbands", 5]]
maximum_dimension optional	Integer	An integer specifying the maximum size of the normalized version of the input file that will be used for analysis. The value refers to either width or height, whichever is greater. When the provided file has any dimension above maximum_dimension, its normalized version is downscaled to the latter, preserving aspect ratio. The normalization also implies standard transformations on color channels and the like. **Example**: 32

Image Transformations

When creating or cloning a single image source, it is possible to specify a list of transformations to be performed on the original image (before normalizing it), using the transformations parameter. This parameter's value must be a list of maps, each one containing two mandatory keys:

- **operation**: a string, the name of the operation to be performed.
- **arguments**: a map, naming the parameter values to be used with the given operation.

For instance:

```json
"transformations": [
  {
    "operation": "crop",
    "arguments": {"x1": 0, "y1": 10, "x2": 100, "y2": 200}
  },
  ...
]
```

Currently supported operations:

- **crop**: parameters are the coordinates of the upper left (`x1` , `y1`) and bottom right (`x2` , `y2`) corners of the cropping region.

Item Analysis

An **item_analysis** object is composed of any combination of the following properties.

Property	Type	Description
limit optional	Integer, default is **10000**	The maximum number of items that will be used for summarization and modeling. **Example**: 1000
pruning_strategy optional	String, default is **"nearest_to_frequency"**	Parameter describing how that pruning is performed. When the number of different items is over the configured limit, we must discard some of them. Available values are **nearest_to_frequency** to keep those items whose frequency is close to a fixed occurrence rate given by the parameter **target_frequency** or **most_frequent** to take the most frequent items up to a total of **limit**. **Example**: "most_frequent"
separator optional	Char, default is **null**	A character used as the item separator. Defaults to null for auto-detect. **Example**: ";"
separator_regexp optional	String, default is **null**	A regular expression used to identify item separators. If provided, it overrides separator. Default to null to use the value for separator. **Example**: ";;"
target_frequency optional	Float, default is **1/3**	A number between 0 and 1 specifying the occurrence rate used when the pruning strategy is **nearest_to_frequency**. **Example**: 0.2

Term Analysis

A **term_analysis** object is composed of any combination of the following properties.

Property	Type	Description
case_sensitive optional	Boolean, default is **false**	Whether text analysis should be case sensitive or not. **Example**: true
enabled optional	Boolean, default is **true**	Whether text processing should be enabled or not. **Example**: true
excluded_terms optional	Array of Strings,, default is **[], an empty list**	Specifies a list of terms to ignore when performing term analysis. **Example**: ["I", "he", "she", "it", "you", "they", "am", "is", "are", "a", "an", "the"]
language optional	String, default is **"en"**	The default language of text fields in a two-letter language code, which will change the resulting stemming and tokenization. Available options are: "ar", "ca", "cs", "da", "de", "en", "es", "fa", "fi", "fr", "hu", "it", "ja", "ko", "nl", "pl", "pt", "ro", "ru", "sv", "tr", "zh", "none", or null for auto-detect. **Example**: "es"
ngrams optional	Integer, default is **1**	A positive integer n that specifies the use of all sequences of consecutive tokens of length n should be considered as terms, in addition to their constituent tokens (when separated by a single space and no stopwords). See n-gram for more information.The minimum value is 1 and maximum value is 5. **Example**: 5
stem_words optional	Boolean, default is **true**	Whether lemmatization (stemming) of terms should be done, according to linguistic rules in the provided language. Note that if the language is, for example, zh, even English words will not be lemmatized as with English rules. **Example**: true
stopword_diligence optional	String, default is **"light"**	The aggressiveness of stopword removal, where the levels are **light**, **normal** or **aggressive**, in order, where each level is a superset of words in the previous ones. The most common languages will add stopwords at each level, but less common languages

Property	Type	Description
		may not. **Example**: "normal"
stopword_removal optional	String, default is **"selected_language"**	A string or keyword specifying the type of stopword removal to perform. Available options are where it can be **none** (remove no stopwords), **selected_language** (remove stopwords from the provided language), and **all_languages** (remove stopwords from all languages). Note that this parameter supersedes **use_stopwords** if provided. Also note that the null language does have a non-empty stopword list such as single numeric digits. **Example**: "all_languages"
term_filters optional	Array of Strings	Filters that should be applied to the chosen terms. Available options are: • **non_dictionary**: Filters all terms that are not dictionary words in the provided language. • **non_language_characters**: Filters any term containing a character not from the Unicode blocks common for words in the provided language. For example, if the language is ru (Russian), all terms containing non-Cyrillic characters will be filtered out. Non-language characters always includes numeric digits, regardless of language. • **numeric_digits**: Filters any term that contains a numeric digit in [0-9]. • **html_keywords**: Filters JavaScript/HTML keywords commonly seen in HTML documents. • **single_tokens**: Filters terms that contain only a single token (e.g., only pass full terms). For this to return a non-empty set of terms, you must specify token_mode as either all or full_terms_only. **Example**: "html_keywords"
		A list of strings specifying regular expressions to be matched against input documents. If present, these

Property	Type	Description
term_regexps optional	Array	regular expressions will automatically be chosen for the final term list, and their per-document occurrence counts will be the number of matches of the expression in that document.
token_mode optional	String, default is **"all"**	Whether **tokens_only**, **full_terms_only** or **all** should be tokenized. **Example**: "tokens_only"
use_stopwords optional	Boolean, default is **true**	Whether to use stop words or not. This fields is deprecated in favor of **stopword_removal**. **Example**: true

Synthetic Object

A **synthetic object** is composed of the following properties.

Property	Type	Description
cat_padding optional	Integer, default is **0**	The number of padding characters to prepend to category names for testing long strings. **Example**: 1
fields	Integer	The number of fields to include in the synthetic source. The generated synthetic source will have as many **fields** as set by the argument fields plus a "class" field. **Example**: 10
frac_cat optional	Number, default is **0.5**	The fraction (between 0 and 1) of attributes that are categorical. **Example**: 0.3
frac_time optional	Number, default is **0**	The fraction (between 0 and 1) of attributes that represent times. **Example**: 0.1
missing optional	Number, default is **0**	The fraction (between 0 and 1) of missing values (null) that occur in the data. **Example**: 0.1

Property	Type	Description
noise optional	Number, default is **0**	The fraction (between 0 and 1) of attributes not correlated with the class attribute. **Example**: 0.1
num_cats optional	Integer, default is **3**	The number of categories for categorical attributes. **Example**: 5
num_classes optional	Integer, default is **3**	The number of classes for the final class attribute. **Example**: 5
rows	Integer	The number of rows to include in the synthetic source. **Example**: 10
sparsity optional	Number, default is **0**	The fraction (between 0 and 1) rows that will have a value of zero for each numeric field. **Example**: 0.1

Text Processing

While the handling of **numeric**, **categorical**, or **items** fields within a decision tree framework is fairly straightforward, the handling of **text** fields can be done in a number of different ways. **BigML.io** takes a basic and reasonably robust approach, leverging some basic NLP techniques along with a simple bag-of-words style method of feature generation.

At the source level, **BigML.io** attempts to do basic language detection. Initially the language can be English (**"en"**), Spanish (**"es"**), Catalan/Valencian (**"ca"**), Dutch (**"nl"**), French (**"fr"**), German (**"de"**), Portuguese (**"pt"**), or **"none"** if no language is detected. In the near future, **BigML.io** will support many more languages.

For **text** fields, **BigML.io** adds potentially five keys to the detected fields, all of which are placed in a map under **term_analysis**.

The first is language, which is mapped to the detected language.

There are also three boolean keys, **case_sensitive**, **use_stopwords**, and **stem_words**. The **case_sensitive** key is false by default. **use_stopwords** should be true if we should include stopwords in the vocabulary for the detected field during text summarization. **stem_words** should be true if **BigML.io** should perform word stemming on this field, which maps forms of the same term to the same key when summarizing or generating models. By default, **use_stopword** is false and **stem_word**s is true for languages other than "none" and

they are not present otherwise.

Finally, **token_mode** determines the tokenization strategy. It may be set as either **tokens_only**, **full_terms_only**, and **all**. When set as **tokens_only** then individual words are used as terms. For example, "ML for all" becomes ["ML", "for", "all"]. However, when **full_terms_only** is selected, then the entire field is treated as a single term as long as it is shorter than 256 characters. In this case "ML for all" stays ["ML for all"]. If **all** is selected, then both full terms and tokenized terms are used. In this case ["ML for all"] becomes ["ML", "for", "all", "ML for all"]. The default for **token_mode** is **all**.

There are a few details to note:

- If **full_terms_only** is selected, then no stemming will occur even if stem_words is true.
- Also, when either all or **tokens_only** are selected, a term must appear at least twice to be selected for the tag cloud. However **full_terms_only** lowers this limit to a single occurrence.
- Finally, if the **language** is "none", or if a language does not have an algorithm available for stopword removal or stemming, the **use_stopwords** and **stem_words** keys will have no effect.

Items Detection

BigML.io automatically detects as **items** fields that have many different categorical values per instance separated by non-alphanumeric characters, so they can't be considered either categorical or text fields

These kind of fields can be found in transactional datasets where each instance is associated to a different set of products contained within one field. For example, datasets containing all products bought by users or prescription datasets where each patient is associated to different treatments. These datasets are commonly used for **Association Discovery** to find relationships between different items.

Find the two CSV examples below that could be considered items fields:

User, Prescription
John Doe, medicine 1; medicine 2
Jane Roe, medicine 1; medicine 3; medicine 4; medicine 6

Transaction, Product
12345, product 1; product 2; product 5; product 6; product 7
67890, product 1; product 3; product 4

In the examples above, the fields **Prescription** and **Products** will be considered as **items** and each different value will be a unique item.

Once a field has been detected as **items**, **BigML** tries to automatically detect which is the best separator for your items. For example, for the following itemset {hot dog; milk, skimmed; chocolate}, the best separator is the semicolon which yields three different items: 'hot dog', 'milk, skimmed' and 'chocolate'.

For **items** fields, there are five different parameters you can configure under the property group item_analysis, which includes separator that allows you to specify which **separator** you want to set for your items.

Note that **items** fields can't be eligible as target fields for models, logistic regression, and ensembles, but they can be used as predictors. For anomaly detection, they can't be included as an input field to calculate the anomaly score, although they can be selected as summary fields.

Datetime Detection

During the source pre-scan **BigML** tries to determine the data type of each field in your file. This process automatically detects datetime fields and, if **disable_datetime** is not explicitly set to "false", BigML will generate additional fields with their components.

For instance, if a field named "date" has been identified as a **datetime** with format "YYYY-MM-dd", four new fields will be automatically added to the source, namely "date.year", "date.month", "date.day-of-month" and "date.day-of-week". For each row, these new fields will be filled in automatically by parsing the value of their parent field, "date". For example, if the latter contains the value "1969-07-14", the autogenerated columns in that row will have the values 1969, 7, 14 and 1 (because that day was Monday). As noted before, autogenaration can be disabled by setting **disable_datetime** option to "true", either in the create source request or later in an update source operation.

When a field is detected as **datetime**, **BigML** tries to determine its format for parsing the values and generate the fields with their components. By default, BigML accepts ISO 8601 time formats (YYYY-MM-DD) as well as a number of other common European and US formats, as seen in the table below:

time_format Name	Example	Format
basic-date-time	19690714T173639.592Z	`YYYYMMdd'T'HHmmss.SSSXX`
basic-date-time-no-ms	19690714T173639Z	`YYYYMMdd'T'HHmmssXX`
basic-iso-date	19690714Z	`YYYYMMddXX`
basic-ordinal-date-time	1969195T173639.592Z	`YYYYDDD'T'HHmmss.SSSXX`
basic-ordinal-date-time-no-ms	1969195T173639Z	`YYYYDDD'T'HHmmssXX`
basic-t-time	T173639.592Z	`'T'HHmmss.SSSXX`
basic-t-time-no-ms	T173639Z	`'T'HHmmssXX`

time_format Name	Example	Format
basic-time	173639.592Z	`HHmmss.SSSXX`
basic-time-no-ms	173639Z	`HHmmssXX`
basic-week-date	1969W291	`xxxx'W'wwe`
basic-week-date-time	1969W291T173639.592Z	`xxxx'W'wwe'T'Hmmss.SSSXX`
basic-week-date-time-no-ms	1969W291T173639Z	`xxxx'W'wwe'T'HmmssXXX`
bigquery	1969-07-14 17:36:39Z	`Y-M-d H:m:sXXX`
bigquery-alt	1969-07-14 17:36:39 UTC	`Y-M-d H:m:s z`
bigquery-alt-millisecond	1969-7-14 17:36:39.592000 UTC	`Y-M-d H:m:s.SSSSSS z`
bigquery-millisecond	1969-7-14 17:36:39.592000Z	`Y-M-d H:m:s.SSSSSSXXX`
clock-minute	5:36 PM	`h:m a`
clock-minute-nospace	5:36PM	`h:ma`
clock-second	5:36:39 PM	`h:m:s a`
clock-second-nospace	5:36:39PM	`h:m:sa`
date	1969-07-14	`Y-M-d`
date-hour	1969-07-14T17	`YYYY-MM-dd'T'H`
date-hour-minute	1969-07-14T17:36	`YYYY-MM-dd'T'H:mm`
date-hour-minute-second	1969-07-14T17:36:39	`YYYY-MM-dd'T'H:mm:ss`

time_format Name	Example	Format
date-hour-minute-second-fraction	1969-07-14T17:36:39.592	`YYYY-MM-dd'T'H:mm:ss.SSS`
date-hour-minute-second-fraction-with-solidus	1969/07/14T17:36:39.592	`YYYY/MM/dd'T'H:mm:ss.SSS`
date-hour-minute-second-ms	1969-07-14T17:36:39.592	`YYYY-MM-dd'T'HH:mm:ss.SSS`
date-hour-minute-second-ms-with-solidus	1969/07/14T17:36:39.592	`YYYY/MM/dd'T'HH:mm:ss.SSS`
date-hour-minute-second-with-solidus	1969/07/14T17:36:39	`YYYY/MM/dd'T'H:mm:ss`
date-hour-minute-with-solidus	1969/07/14T17:36	`YYYY/MM/dd'T'H:mm`
date-hour-with-solidus	1969/07/14T17	`YYYY/MM/dd'T'H`
date-time	1969-07-14T17:36:39.592Z	`YYYY-MM-dd'T'H:mm:ss.SSSXXX`
date-time-no-ms	1969-07-14T17:36:39Z	`YYYY-MM-dd'T'H:mm:ssXXX`
date-time-no-ms-with-solidus	1969/07/14T17:36:39Z	`YYYY/MM/dd'T'H:mm:ssXXX`
date-time-with-solidus	1969/07/14T17:36:39.592Z	`YYYY/MM/dd'T'H:mm:ss.SSSXXX`
date-with-solidus	1969/07/14	`YYYY/MM/dd`
elasticsearch-nanos	1969-07-14T17:36:39.592000Z	`YYYY-MM-dd'T'HH:mm:ss.SSSSSSZ`
eu-date	14/7/1969	`d/M/Y`
eu-date-clock-minute	14/7/1969 5:36 PM	`d/M/Y h:m a`
eu-date-clock-minute-nospace	14/7/1969 5:36PM	`d/M/Y h:ma`

time_format Name	Example	Format
eu-date-clock-second	14/7/1969 5:36:39 PM	`d/M/Y h:m:s a`
eu-date-clock-second-nospace	14/7/1969 5:36:39PM	`d/M/Y h:m:sa`
eu-date-millisecond	14/7/1969 17:36:39.592	`d/M/Y H:m:s.SSS`
eu-date-minute	14/7/1969 17:36	`d/M/Y H:m`
eu-date-second	14/7/1969 17:36:39	`d/M/Y H:m:s`
eu-ddate	14.7.1969	`d.M.Y`
eu-ddate-clock-minute	14.7.1969 5:36 PM	`d.M.Y h:m a`
eu-ddate-clock-minute-nospace	14.7.1969 5:36PM	`d.M.Y h:ma`
eu-ddate-clock-second	14.7.1969 5:36:39 PM	`d.M.Y h:m:s a`
eu-ddate-clock-second-nospace	14.7.1969 5:36:39PM	`d.M.Y h:m:sa`
eu-ddate-millisecond	14.7.1969 17:36:39.592	`d.M.Y H:m:s.SSS`
eu-ddate-minute	14.7.1969 17:36	`d.M.Y H:m`
eu-ddate-second	14.7.1969 17:36:39	`d.M.Y H:m:s`
eu-sdate	14-7-1969	`d-M-Y`
eu-sdate-clock-minute	14-7-1969 5:36 PM	`d-M-Y h:m a`
eu-sdate-clock-minute-nospace	14-7-1969 5:36PM	`d-M-Y h:ma`
eu-sdate-clock-second	14-7-1969 5:36:39 PM	`d-M-Y h:m:s a`
eu-sdate-clock-second-nospace	14-7-1969 5:36:39PM	`d-M-Y h:m:sa`

time_format Name	Example	Format
eu-sdate-millisecond	14-7-1969 17:36:39.592	`d-M-Y H:m:s.SSS`
eu-sdate-minute	14-7-1969 17:36	`d-M-Y H:m`
eu-sdate-second	14-7-1969 17:36:39	`d-M-Y H:m:s`
hour-minute	17:36	`H:mm`
hour-minute-second	17:36:39	`H:mm:ss`
hour-minute-second-fraction	17:36:39.592	`H:mm:ss.SSS`
hour-minute-second-ms	17:36:39.592	`H:mm:ss.SSS`
iso-date	1969-07-14Z	`Y-M-dXXX`
iso-date-time	1969-07-14T17:36:39.592Z	`Y-M-d'T'HH:mm:ss.SSSXXX`
iso-instant	1969-07-14T17:36:39.592Z	`Y-M-d'T'HH:mm:ss.SSSXXX`
iso-local-date	1969-07-14	`Y-M-d`
iso-local-date-time	1969-07-14T17:36:39.592	`Y-M-d'T'HH:mm:ss`
iso-local-time	17:36:39.592	`HH:mm:ss.SSS`
iso-offset-date	1969-07-14Z	`Y-M-dXXX`
iso-offset-date-time	1969-07-14T17:36:39.592Z	`Y-M-d'T'HH:mm:ss.SSSXXX`
iso-offset-time	17:36:39.592Z	`HH:mm:ss.SSSXXX`
iso-ordinal-date	1969-195Z	`YYYY-DDDXX`
iso-time	17:36:39.592Z	`HH:mm:ss.SSSXXX`

time_format Name	Example	Format
iso-week-date	1969-W29-1Z	`xxxx-'W'ww-ez`
iso-zoned-date-time	1969-07-14T17:36:39.592Z	`Y-M-d'T'HH:mm:ss.SSSXXX`
mysql	1969-07-14 17:36:39	`YYYY-MM-dd H:mm:ss`
no-t-date-hour-minute	1969-7-14 17:36	`YYYY-MM-dd H:m`
odata-format	/Date(-14711000408)/	
ordinal-date-time	1969-195T17:36:39.592Z	`YYYY-DDD'T'H:mm:ss.SSSXXX`
ordinal-date-time-no-ms	1969-195T17:36:39Z	`YYYY-DDD'T'H:mm:ssXXX`
rfc-1123-date-time	Mon, 14 Jul 1969 17:36:39 GMT	`EEE, dd MMM YYYY HH:mm:ss z`
rfc822	Mon, 14 Jul 1969 17:36:39 +0000	`EEE, dd MMM YYYY HH:mm:ss ZZZZ`
t-time	T17:36:39.592Z	`'T'HH:mm:ss.SSSXXX`
t-time-no-ms	T17:36:39Z	`'T'HH:mm:ssXXX`
time	17:36:39.592Z	`HH:mm:ss.SSSXXX`
time-no-ms	17:36:39Z	`HH:mm:ssXXX`
timestamp	-14711000	
timestamp-msecs	-14711000408	
twitter-time	Mon Jul 14 17:36:39 +0000 1969	`E MMM d H:m:s Z Y`
twitter-time-alt	1969-7-14 17:36:39 +0000	`Y-M-d H:m:s Z`

time_format Name	Example	Format
twitter-time-alt-2	1969-7-14 17:36 +0000	`Y-M-d H:m Z`
twitter-time-alt-3	Mon Jul 14 17:36 +0000 1969	`E MMM d H:m Z Y`
us-date	7/14/1969	`M/d/Y`
us-date-clock-minute	7/14/1969 5:36 PM	`M/d/Y h:m a`
us-date-clock-minute-nospace	7/14/1969 5:36PM	`M/d/Y h:ma`
us-date-clock-second	7/14/1969 5:36:39 PM	`M/d/Y h:m:s a`
us-date-clock-second-nospace	7/14/1969 5:36:39PM	`M/d/Y h:m:sa`
us-date-millisecond	7/14/1969 17:36:39.592	`M/d/Y H:m:s.SSS`
us-date-minute	7/14/1969 17:36	`M/d/Y H:m`
us-date-second	7/14/1969 17:36:39	`M/d/Y H:m:s`
us-sdate	7-14-1969	`M-d-Y`
us-sdate-clock-minute	7-14-1969 5:36 PM	`M-d-Y h:m a`
us-sdate-clock-minute-nospace	7-14-1969 5:36PM	`M-d-Y h:ma`
us-sdate-clock-second	7-14-1969 5:36:39 PM	`M-d-Y h:m:s a`
us-sdate-clock-second-nospace	7-14-1969 5:36:39PM	`M-d-Y h:m:sa`
us-sdate-millisecond	7-14-1969 17:36:39.592	`M-d-Y H:m:s.SSS`
us-sdate-minute	7-14-1969 17:36	`M-d-Y H:m`
us-sdate-second	7-14-1969 17:36:39	`M-d-Y H:m:s`

time_format Name	Example	Format
week-date	1969-W29-1	`xxxx-'W'ww-e`
week-date-time	1969-W29-1T17:36:39.592Z	`xxxx-'W'ww-e'T'H:mm:ss.SSSXXX`
week-date-time-no-ms	1969-W29-1T17:36:39Z	`xxxx-'W'ww-e'T'H:mm:ssXXX`
weekyear-week	1969-W29	`xxxx-'W'ww`
weekyear-week-day	1969-W29-1	`xxxx-'W'ww-e`
year-month	1969-07	`YYYY-MM`
year-month-day	1969-07-14	`YYYY-MM-dd`

It might happen that BigML is not able to determine the right format of your datetime field. In that case, it will be considered either a **text** or a **categorical** field. You can override that assignment by setting the **optype** of the field to **datetime** and passing the appropriate format in **time_formats**. For instance:

curl

```
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{"fields": {"000004": {"optype": "datetime", "time_formats": ["date"]}}}'
```

If none of the predefined formats in the table above meets your needs, you can set your own format string using the Java DateTimeFormatter specification for datetime patterns. There are examples available in the column *Format* of the table above.

All letters 'A' to 'Z' and 'a' to 'z' are reserved as pattern letters. The following pattern letters are defined:

Symbol	Meaning	Presentation	Examples
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04

Symbol	Meaning	Presentation	Examples
y	year-of-era	year	2004; 04
D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10
g	modified-julian-day	number	2451334
Q/q	quarter-of-year	number/text	3; 03; Q3; 3rd quarter
Y	week-based-year	year	1996; 96
w	week-of-week-based-year	number	27
W	week-of-month	number	4
E	day-of-week	text	Tue; Tuesday; T
e/c	localized day-of-week	number/text	2; 02; Tue; Tuesday; T
F	day-of-week-in-month	number	3
a	am-pm-of-day	text	PM
h	clock-hour-of-am-pm (1-12)	number	12
K	hour-of-am-pm (0-11)	number	0
k	clock-hour-of-day (1-24)	number	24
H	hour-of-day (0-23)	number	0
m	minute-of-hour	number	30
s	second-of-minute	number	55

Symbol	Meaning	Presentation	Examples
S	fraction-of-second	fraction	978
A	milli-of-day	number	1234
n	nano-of-second	number	987654321
N	nano-of-day	number	1234000000
V	time-zone ID	zone-id	America/Los_Angeles; Z; -08:30
v	generic time-zone name	zone-name	Pacific Time; PT
z	time-zone name	zone-name	Pacific Standard Time; PST
O	localized zone-offset	offset-O	GMT+8; GMT+08:00; UTC-08:00
X	zone-offset 'Z' for zero	offset-X	Z; -08; -0830; -08:30; -083015; -08:30:15
x	zone-offset	offset-x	+0000; -08; -0830; -08:30; -083015; -08:30:15
Z	zone-offset	offset-Z	+0000; -0800; -08:00
p	pad next	pad modifier	1
'	escape for text	delimiter	
''	single quote	literal	'
[	optional section start		
]	optional section end		
#	reserved for future use		
{	reserved for future use		

Symbol	Meaning	Presentation	Examples
}	reserved for future use		

The count of pattern letters determines the format.

Text: If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. Thus, "EEEE" might output "Monday" whereas "E" might output "Mon" (the short form of Monday).

Number: The minimum number of digits. Shorter numbers are zero-padded to this amount. Thus, "HH" might output "09" whereas "H" might output "9" (for the hour-of-day of 9 in the morning).

Year: Numeric presentation for year and weekyear fields are handled specially. For example, if the count of 'y' is 2, the year will be displayed as the zero-based year of the century, which is two digits.

Month: 3 or over, use text, otherwise use number. Thus, "MM" might output "03" whereas "MMM" might output "Mar" (the short form of March) and "MMMM" might output "March".

Zone: 'Z' outputs offset without a colon, 'ZZ' outputs the offset with a colon, 'ZZZ' or more outputs the zone id.

Zone names: Time zone names ('z') cannot be parsed.

Any unrecognized letter is an error. Any non-letter character, other than '[', ']', '{', '}', '#' and the single quote will be output directly. Despite this, it is recommended to use single quotes around all characters that you want to output directly to ensure that future changes do not break your application.

```curl
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH" \
    -X PUT \
    -H 'content-type: application/json' \
    -d '{"fields": {"000004": {"optype": "datetime", "time_formats": ["YYYY-MM-dd"]}}}'
```

Retrieving a Source

Each **source** has a unique identifier in the form **"source/id"** where id is a string of 24 alpha-numeric characters that you can use to retrieve the **source**.

To retrieve a **source** with **curl**:

```curl
```

```
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH"
```

You can also use your browser to visualize the **source** using the full **BigML.io** URL or pasting the **source/id** into the BigML labs dashboard.

Source Properties

Once a **source** has been successfully created it will have the following properties.

Property	Type	Description
category filterable, sortable, updatable	Integer	One of the categories in the table of categories that help classify this resource according to the domain of application.
closed updatable	Boolean	Whether the **source** is closed or open.
code	Integer	One of the HTTP status code. This will be 201 upon successful creation of the **source** and 200 afterwards. Make sure that you check the code that comes with the status attribute to make sure that the **source** creation has been completed without errors.
content_type filterable, sortable	String	This is the MIME content-type as provided by your HTTP client. The content-type can help **BigML.io** to better parse your file. For example, if you use **curl**, you can alter it using the type option "-F file=@iris.csv;type=text/csv".
created filterable, sortable	ISO-8601 Datetime	This is the date and time in which the **source** was created with microsecond precision. It follows this pattern yyyy-MM-ddThh:mm:ss.SSSSSS. All times are provided in Coordinated Universal Time (UTC).
creator	String	The user that created the **source**.
description updatable	String	A text describing the **source**. It can contain restricted markdown to decorate the text.

Property	Type	Description
disable_datetime updatable	Boolean	**False** when BigML didn't generate new fields from existing date-time fields.
execution_id filterable, sortable	String	The **execution id** that built the **source**.
execution_status filterable, sortable	Boolean	Whether the **execution** is still available or has been deleted.
field_types	Object	A dictionary with an entry per field type in your data and the total number of fields of that type. **Example**: { "auto_generated": {}, "categorical": 1, "datetime": 0, "image": 0, "items": 0, "numeric": 8, "path": 0, "regions": 0, "text": 0, "total": 9 }
fields updatable	Object	A dictionary with an entry per field (column) in your data. Each entry includes the column number, the name of the field, the type of the field, a specific locale if it differs from the source's one, and specific missing tokens if the differ from the source's one. This property is very handy to update sources according to your own parsing preferences. **Example**: { "000001": { "name": "length_1", "optype": "numeric", "locale": "es-ES", "missing_tokens": ["x"] }, "000003": { "name": "length_2",

Property	Type	Description
		"optype": "numeric", "locale": "en-US", "missing_tokens": ["na"] } }
fields_meta	Object	A dictionary with meta information about the fields dictionary. It specifies the **total** number of fields, the current **offset**, and **limit**, and the number of fields (**count**) returned.
fields_preview	Object	A dictionary with an entry per field (column) in your data. Each entry includes a list of values of that field in a reduced set of instances.
file_name filterable, sortable	String	The name of the file as you submitted it.
format filterable, sortable	String	The format of the source.
height	Integer	The height, in pixels, of the normalized version of the image.
image_analysis updatable	Object	Set of parameters to analysis performed on image data for the **source**.
md5	String	The file MD5 Message-Digest Algorithm as specified by RFC 1321.
name filterable, sortable, updatable	String	The name of the **source** as your provided or the name of the file by default.
name_options filterable, sortable	String	Information about the **source**.
		An array of dictionaries. Each entry represents a label and can include all available field description properties (such as name, optype, description, term_analysis, item_analysis) and, in addition, "value", with the default value of each label field for

Property	Type	Description
new_fields updatable	Array of Objects	this image. All optypes except image are allowed. **Example**: [{ "label": "color", "name": "color", "optype": "categorical", "description": "background color", "value": "antiquewhite" }]
number_of_anomalies filterable, sortable	Integer	The current number of **anomalies** that use this **source**.
number_of_anomalyscores filterable, sortable	Integer	The current number of **anomaly scores** that use this **source**.
number_of_associations filterable, sortable	Integer	The current number of **associations** that use this **source**.
number_of_associationsets filterable, sortable	Integer	The current number of **association sets** that use this **source**.
number_of_centroids filterable, sortable	Integer	The current number of **centroids** that use this **source**.
number_of_clusters filterable, sortable	Integer	The current number of **clusters** that use this **source**.
number_of_correlations filterable, sortable	Integer	The current number of **correlations** that use this **source**.
number_of_datasets filterable, sortable	Integer	The current number of **datasets** that use this **source**.
number_of_ensembles filterable, sortable	Integer	The current number of **ensembles** that use this **source**.
number_of_forecasts filterable, sortable	Integer	The current number of **forecasts** that use this **source**.

Property	Type	Description
number_of_linearregressions filterable, sortable	Integer	The current number of **linear regressions** that use this **source**.
number_of_logisticregressions filterable, sortable	Integer	The current number of **logistic regressions** that use this **source**.
number_of_models filterable, sortable	Integer	The current number of **models** that use this **source**.
number_of_optimls filterable, sortable	Integer	The current number of **optimls** that use this **source**.
number_of_pca filterable, sortable	Integer	The current number of **pcas** that use this **source**.
number_of_predictions filterable, sortable	Integer	The current number of **predictions** that use this **source**.
number_of_statisticaltests filterable, sortable	Integer	The current number of **statistical tests** that use this **source**.
number_of_timeseries filterable, sortable	Integer	The current number of **timeseries** that use this **source**.
number_of_topicdistributions filterable, sortable	Integer	The current number of **topic distributions ** that use this **source**.
number_of_topicmodels filterable, sortable	Integer	The current number of **topic models** that use this **source**.
origin filterable, sortable	String	The **source/id** of the original source.
original_height	Integer	The height of the image before normalization.
original_size	Integer	The size of the image before normalization.
original_width	Integer	The width of the uploaded image before normalization.

Property	Type	Description
parent_sources filterable, sortable	Array	The list of **ids** of sources for which this source is a component.
private filterable, sortable, updatable	Boolean	Whether the **source** is public or not.
project filterable, sortable, updatable	String	The **project/id** the resource belongs to.
remote	String	URL of the remote data source.
resource	String	The **source/id**.
shared filterable, sortable, updatable	Boolean	Whether the **source** is shared using a private link or not.
shared_clonable filterable, sortable, updatable	Boolean	Whether the shared **source** can be cloned or not.
shared_hash	String	The hash that gives access to this **source** if it has been shared using a private link.
sharing_key	String	The alternative key that gives read access to this **source**.
size filterable, sortable	Integer	The number of bytes of the **source**.
source_csv	String	For a composite source with format csv+image, the component with format csv.
source_parser updatable	Object	Set of parameters to parse the **source**.
sources filterable, sortable, updatable	Array	The list of **ids** of component sources.
status	Object	A description of the status of the **source**. It includes a code, a message, and some extra information. See

Property	Type	Description
		the table below.
subscription filterable, sortable	Boolean	Whether the **source** was created using a subscription plan or not.
synthetic	Object	Set of parameters to generate a synthetic **source** presumably for activities such as testing, prototyping and benchmarking.
tags filterable, updatable	Array of Strings	A list of user tags that can help classify and index this resource.
term_analysis updatable	Object	Set of parameters that define how text analysis should work for text fields.
type filterable, sortable	Integer	The type of **source**. • **0** if the source was created using a local file. • **1** if the source has been created using a remote URL • **2** if the source has been created using inline data. • **3** if the source has been created using synthetically generated data. • **4** if the source has been created from an external data repository (rbdms, elasticsearch, ...) • **5** if source is a composite source
updated filterable, sortable	ISO-8601 Datetime	This is the date and time in which the **source** was updated with microsecond precision. It follows this pattern yyyy-MM-ddThh:mm:ss.SSSSSS. All times are provided in Coordinated Universal Time (UTC).
webhook	Object	A webhook url and an optional secret phrase. See the Section on Webhooks for more details.
width	Integer	The width, in pixels, of the normalized version of the image.

Source Fields

The property **fields** is a dictionary keyed by an auto-generated **id** per each field in the **source**. Each field has as a value an object with the following properties:

Property	Type	Description
column_number	Integer	Specifies the number of column in the original file.
description	String	An even longer description for the field.
item_analysis	Object	Specifies the item analysis parameters for this items field. **Example**: {"separator": ";", "limit": 100, "pruning_strategy": "most_frequent", "target_frequency": 0.5}
label	String	A longer and more descriptive name of the field. **Example**: "Sepal length in cm"
locale	String, default is **source's locale**	The specific locale for this field. **Example**: "en-US"
missing_tokens	Array, default is **source's missing tokens**	The specific missing tokens for this field. **Example**: ["NA", "N/A"]
name	String	Name of the column if provided in the header of the source or a name generated automatically otherwise. **Example**: "Sepal length"
optype	String	Specifies the type of the field. It can be **numeric**, **categorical**, **text**, **items**, **image**, **path** or **regions**. **Example**: "text"
term_analysis	Object	Specifies the text analysis parameters for this text field. **Example**: {"enabled": true, "case_sensitive": false, "stem_words": true, "ngrams": 5}

For fields classified with **optype "text"**, the default values specified in the **term_analysis** at the top-level of the source are used.

Non-provided flags by **term_analysis** take their default value, i.e., false for booleans, none for **language**.

Besides these global default values, which apply to all text fields (and potential text fields, such as categorical ones that might overflow to text during dataset creation), it's possible to specify **term_analysis** flags on a per-field basis.

For fields classified with **optype "items"**, the default values specified in the **item_analysis** at the top-level of the source are used.

Like **term_analysis**, non-provided flags by **item_analysis** take their default value and it's possible to specify **item_analysis** flags on a per-field basis as well at the global level, too.

Source Status

Before a **source** is successfully created, **BigML.io** makes sure that it has been uploaded in an understandable format, that the data that it contains is parseable, and that the types for each column in the data can be inferred successfully. The **source** goes through a number of states until all these analyses are completed. Through the status field in the **source** you can determine when the **source** has been fully processed and is ready to be used to create a dataset. These are the fields that a **source's status** has:

Property	Type	Description
code	Integer	A status code that reflects the status of the **resource** creation. It can be any of those that are explained here.
elapsed	Integer	Number of milliseconds that **BigML.io** took to process the **resource**.
message	String	A human readable message explaining the status.
progress	Float, between 0 and 1	How far **BigML.io** has progressed building the **resource**.

Once a **source** has been successfully created, it will look like:

```json
{
    "category": 0,
    "charset": "UTF-8",
    "closed": false,
    "code": 200,
    "configuration": null,
    "configuration_status": false,
```

```
    "content_type": "text/plain;UTF-8",
    "created": "2021-03-02T11:22:23.230000",
    "creator": "alfred",
    "description": "",
    "disable_autolabel": false,
    "disable_datetime": false,
    "field_types": {
```


Filtering and Paginating Fields from a Source

A **source** might be composed of hundreds or even thousands of fields. Thus when retrieving a **source**, it's possible to specify that only a subset of fields be retrieved, by using any combination of the following parameters in the query string (unrecognized parameters are ignored):

Parameter	Type	Description
fields optional	Comma-separated list	A comma-separated list of **field** IDs to retrieve. **Example**: "fields=000000,000002"
full optional	Boolean	If false, no information about fields is returned. **Example**: "full=false"
iprefix optional	String	A case-insensitive string to retrieve fields whose name start with the given prefix; It is possible to specify more than one iprefix by repeating the parameter, in which case the union of the results is returned. **Example**: "iprefix=INCOME"
limit optional	Integer	Maximum number of **fields** that you will get in the **fields** field. **Example**: "limit=100"
offset optional	Integer	How far off from the first **field** in your **dataset** is the first **field** in the **fields** field. **Example**: "offset=100"
order_by optional	String	Sorting criteria; possible values are **"count"**, **"max"**, **"min"**, **"name"**, and **"type"**, and their negated values (**"-count"**, **"-name"**, etc.) to specify a descending order. **Example**: "order_by=name"

Parameter	Type	Description
prefix optional	String	A case-sensitive string to retrieve fields whose name start with the given prefix; It is possible to specify more than one prefix by repeating the parameter, in which case the union of the results is returned. **Example**: "prefix=income"

Since **fields** is a map and therefore not ordered, the returned fields contain an additional key, **order**, whose integer (increasing) value gives you their ordering. In all other respects, the source is the same as the one you would get without any filtering parameter above.

The **fields_meta** field can help you paginate fields. Its structure is as follows:

Property	Type	Description
count optional	Integer	Specifies the current number of fields in the resource.
limit optional	Integer	The maximum number of fields that will be returned in the resource.
offset optional	Integer	The current offset in the pagination of fields.
total optional	Integer	The total number of fields in the resource.

Note that paginating fields might only be worth if you are going to deal with really wide (i.e., more than 200 fields).

Updating a Source

To update a **source**, you need to PUT an object containing the fields that you want to update to the **source**'s base URL. The content-type must always be: **"application/json"**. If the request succeeds, **BigML.io** will return with an HTTP 202 response with the updated **source**.

For example, to update a **source** with a new name and a new locale you can use **curl** like this:

curl

```
curl "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH" \
```

```
    -X PUT \
     -H 'content-type: application/json' \
     -d '{"name": "a new name", "source_parser": {"locale": "es-ES"}}'
```

See this section for more details.

Deleting a Source

To delete a **source**, you need to issue a HTTP DELETE request to the **source/id** to be deleted.

Using **curl** you can do something like this to delete a **source**:

curl

```
curl -X DELETE "https://bigml.io/andromeda/source/603e1fef1f386f43db000000?$BIGML_AUTH"
```

If the request succeeds you will not see anything on the command line unless you executed the command in verbose mode. Successful DELETEs will return **"204 no content"** responses with no body.

Once you delete a **source**, it is permanently deleted. That is, a delete request cannot be undone. If you try to delete a **source** a second time, or a **source** that does not exist, you will receive a **"404 not found"** response.

However, if you try to delete a **source** that is being used at the moment, then **BigML.io** will not accept the request and will respond with a **"400 bad request"** response.

See this section for more details.

Listing Sources

To list all the **source**, you can use the **source** base URL. By default, only the 20 most recent **sources** will be returned. You can see below how to change this number using the **limit** parameter.

You can get your list of **sources** using **curl**.

curl

```
curl "https://bigml.io/andromeda/source?$BIGML_AUTH"
```

See this section for more details. You can also paginate, filter, and order your **sources**.