Databricks Cloud Platform Native REST API 1.2

Overview

This document describes the Databricks Native API that can be used by third party applications to interact with the Spark clusters managed by Databricks Cloud. This is an API similar to the one used by the Databricks Workspace (i.e., notebooks and dashboards) to interact with Spark and offers a tight integration of third party applications with the Databricks Workspace: the same tables (cached or not) can be shared between the third party applications and the Databricks Workspace.

The API described in this document covers several aspects:

- *Cluster management*: describing and selecting existing clusters
- *Execution contexts*: creating unique variable namespaces where Spark commands can be called (in Scala or Python)
- *Command execution*: executing commands within a specific execution context
- *Libraries*: allows uploading 3rd party libraries that can be used in the submitted commands.

This REST API runs over HTTPS. For retrieving information, HTTP GET is used, for state modification, HTTP POST is used. Response content type is json. For POST request, multipart/form-data is used for file upload, otherwise application/x-www-form-urlencoded is used.

In version 1.2, basic authentication is used to authenticate the user for every API call. User's credentials are base64 encoded and are in the HTTP header for every API call, for example "Authorization: Basic YWRtaW46YWRtaW4=".

Next, we present a list of all the API calls and the we show examples and more details on each of the API calls.

Update History

version 1.0, Dec 2014

- First version

version 1.1, March 2015

- Library management is updated

    - Uploaded libraries are visible in UI
    - Persistent library across cluster restart
    - Option to attach a library to new clusters
    - Attach and detach a library to clusters
- Added ability to create, resize, and delete spark clusters

version 1.2, Aug 2015

- Cluster management is updated
    - Use num of worker containers when creating or resizing clusters

List of API calls

Cluster Management

- *https://dbc_api_url/api/1.2/clusters/list* -- lists all spark clusters, including id, name, state
- *https://dbc_api_url/api/1.2/clusters/status* -- retrieves information about a single spark cluster
- *https://dbc_api_url/api/1.2/clusters/restart* -- restart a spark cluster
- *https://dbc_api_url/api/1.2/clusters/create* -- create a new spark cluster
- *https://dbc_api_url/api/1.2/clusters/resize* -- add or remove memory from an existing spark cluster
- *https://dbc_api_url/api/1.2/clusters/delete* -- delete a spark cluster

Execution Context

- *https://dbc_api_url/api/1.2/contexts/create* -- creates an execution context on a specified cluster for a given programming language
- *https://dbc_api_url/api/1.2/contexts/status* -- shows the status of an existing execution context
- *https://dbc_api_url/api/1.2/contexts/destroy* -- destroys an execution context

Command Execution

- *https://dbc_api_url/api/1.2/commands/execute* -- runs a given command
- *https://dbc_api_url/api/1.2/commands/cancel* -- cancels one command
- *https://dbc_api_url/api/1.2/commands/status* -- shows one command's status or result

Known limitations: command execution does not support %run.

Library Upload

- *https://dbc_api_url/api/1.2/libraries/list* -- shows all uploaded libraries
- *https://dbc_api_url/api/1.2/libraries/status* -- shows library statuses
- *https://dbc_api_url/api/1.2/libraries/upload* -- uploads a java-jar, python-egg or python pypi library file
- *https://dbc_api_url/api/1.2/libraries/delete* -- delete a library

- *https://dbc_api_url/api/1.2/libraries/attach* -- schedule an action to attach an uploaded library to a cluster or all clusters
- *https://dbc_api_url/api/1.2/libraries/detach* -- schedule an action to detach a library from a cluster or all clusters

Note: the library API is more experimental than the rest of the API and could suffer bigger changes in the future versions of the API. The behavior is undefined if two libraries containing the same class file are added.

Error Handling

If the API call is successful the returned HTTP status code is 200. In case of internal errors or bad parameters, the returned code is 500 and the error messages also contains a json error message. For example: 500, {"error": "RuntimeException: Unknown cluster id 1"} or 500, {"error": "NumberFormatException: For input string: None"}

For library manager, the API checks the request parameter, and return the following errors

- unknown library id error
  {"error":"NoSuchElementException: key not found: 2915"}
- unknown folder error

Libraries with same name in the same folder are allowed. If two libraries with the same class are attached to the same cluster, the behavior is undefined. Updating a library jar has to delete the library first. Uploading a bad library still returns an id. User can find out the error by status call.

API Call Examples and Details

We now describe examples for each of the existing API calls and describe their parameters.

Cluster Management

Request:
GET
https://dbc_api_url/api/1.2/clusters/list
Reponse:

```
[
    {
        "status": "Running",
        "id": "peaceJam",
        "name": "test",
        "driverIp": "10.12.34.56",
        "jdbcPort": "10000"
    },
```

```
    {
        "status": "Running",
        "id": "peaceJam2",
        "name": "test2",
        "driverIp": "10.12.34.56",
        "jdbcPort": "10000"
    }
]
```

The cluster status is one of “Pending”, “Running”, “Reconfiguring”, “Terminating”, “Terminated”, or “Error”.

Request:
GET
https://dbc_api_url/api/1.2/clusters/status?clusterId=peaceJam
Response:

```
{
  “id”: “peaceJam”,
  “name”: “spark-it-up”,
  “driverIp”: “12.34.56.78”,
  “jdbcPort”: “10000”
}
```

Request:
POST
https://dbc_api_url/api/1.2/clusters/restart
data = {“clusterId”: “peaceJam”}
Response:

```
{
  “id”: “peaceJam”
}
```

Restart the given spark cluster given its id.

Request:
POST
https://dbc_api_url/api/1.2/clusters/create
data = {“name”: “spark-it-up”, “numWorkerContainers”: 6, “useSpot: true”, “sparkVersion”: “1.3.x”, “zoneId”: “us-west-1c”, “sparkConf”: { "spark.speculation" : true }}
Response:

```
{
  “id”: “peaceJam”
}
```

The cluster will enter a “Pending” and then “Running” state. The id returned in the response can be used in future calls to resize or delete a cluster. Optional SparkVersion is one of “1.3.x”, “1.4.x”, “1.5.x”.

Request:
POST
https://dbc_api_url/api/1.2/clusters/resize
data = {“clusterId”: “peaceJam”, “numWorkerContainers”: 6}
Response:

```
{
  “id”: “peaceJam”
}
```

The response is returned immediately. Note that the actual operation may take a while if we need to acquire more instances. The cluster will enter a “Reconfiguring” state and then return to “Running” once the new capacity has been added.

Request:
POST
https://dbc_api_url/api/1.2/clusters/delete
data = {“clusterId”: “peaceJam”}
Response:

```
{
  “id”: “peaceJam”
}
```

The cluster will enter a “Terminating” state before transitioning to “Terminated”.

Execution Context

Request:
POST
https://dbc_api_url/api/1.2/contexts/create
data = {"language": "scala", "clusterId": "peaceJam"}
Reponse:

```
{
  "id": "1793653964133248955"
}
```

The two parameters to this call are the programming language, which can be one of [“scala”, “python”, “sql”] and clusterId, which must be obtained through the cluster manager API.

Request:
GET
https://dbc_api_url/api/1.2/contexts/status?clusterId=peaceJam&contextId=1793653964133248955
Reponse:

```
{
    "status": "Running",
```

```
    "id": "1793653964133248955"
}
```
If the contextId does not exist, http 500 is returned with a message:
```
{
    "error": "ContextNotFound"
}
```

status is one of ["Pending", "Running", "Error"]. The first status represents the execution context is warming up, the second means that commands can be submitted to the execution context while the last state represent an error has occured while creating the execution status.

Request:
POST
https://dbc_api_url/api/1.2/contexts/destroy
data = {"contextId" : "1793653964133248955", "clusterId" : "peaceJam"}
Reponse:
```
{
    "id": "1793653964133248955"
}
```
This API always echo the id back even if it is not a valid contextId.

Command Execution

Request:
POST
https://dbc_api_url/api/1.2/commands/execute
data = {"language": "scala", "clusterId": "peaceJam", "contextId" : "5456852751451433082",
"command": "sc.parallelize(1 to 10).collect"}
Response:
```
{
    "id": "5220029674192230006"
}
```

The response contains the ID of the command being executed.

Note that the language parameter, specifies the language of the command. Currently, a “scala” execution context supports “scala” and “sql” commands, a “python” execution context supports “python” and “sql” commands, while a “sql” execution context supports only “sql” commands.

Request:
POST with a command file
https://dbc_api_url/api/1.2/commands/execute
data = {"language": "sql", "clusterId": "peaceJam", "contextId" : "5456852751451433082"}
files = {"command": "./wiki.sql"}
Response:
```
{
```

```
    "id": "2245426871786618466"
}
```


Request:

GET

https://dbc_api_url/api/1.2/commands/status?clusterId=peaceJam&contextId=5456852751451433082&commandId=5220029674192230006

Example Responses:

1. Text Success Results


```
{
    "status": "Finished",
    "id": "5220029674192230006",
    "results": {
        "resultType": "text",
        "data": "res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)"
    }
}
```


2. Table Success results:


```
{
    "status": "Finished",
    "id": "2245426871786618466",
    "results": {
        "resultType": "table",
        "truncated": "False",
        "data": [
            [
                1847917,
                "& Yet & Yet"
            ],
            [
                28894499,
                "'A morte 'e Carnevale"
            ],
            [
                16496086,
                "'Ayim"
            ],
            [
                2231045,
                "'Hours...'"
            ],
            [
                4410030,
                "'Sgudi 'Snaysi"
            ]
        ],
        "schema": [
            {
                "type": "int",
```

```
                "name": "id"
            },
            {
                "type": "string",
                "name": "title"
            }
        ]
    }
}
```

3. Command is still running:

```
{
    "status": "Running",
    "id": "8370632921745693416"
}
```

4. Finished with errors:

```
{
    "status": "Finished",
    "id": "3881634436378952468",
    "results": {
        "resultType": "error",
        "cause": "InnerExecutionException"
    }
}
```

status can be ["Queued", "Running", "Cancelling", "Finished", "Cancelled", "Error"]. The data field will be present for commands with status: Finished and Cancelled. Note that in this version the API (1.2), command results can only be retrieved once, i.e., after the result has been returned, subsequent commands for status will return an error.

Table results are returned when the result of the command is a (part of a) table, e.g., “select * from table1”.

In version 1.2, the API does not support direct retrieval of binary data, the data in a text result will be encoded using UTF-8. However, 3rd party application developers can easily implement a shim layer to encode/decode binary data inside their commands/application (e.g., using base64 encoding).

Request:
POST
https://dbc_api_url/api/1.2/commands/cancel
data = {"clusterId": "peaceJam", "contextId" : "5456852751451433082", "commandId" : "2245426871786618466"}
Response:
{

```
    "id": "2245426871786618466"
}
```

Library Management

Request:
GET
https://dbc_api_url/api/1.2/libraries/list
Response:

```
[
    {
        "id": "1234",
        "name": "mylib",
        "folder": "/"
    },
    {
        "id": "5678",
        "name": "jar1_2.10-1.0.jar",
        "folder": "/"
    }
]
```

Request:
GET
https://dbc_api_url/api/1.2/libraries/status?libraryId=1234
Response:

```
    {
        "id": "1234",
        "name": "mytestjar",
        "folder": "/",
        "files": ["7fbcf1de_3a36_4628_8173_172413292907-jar4_2.10_1.0.jar"],
        "libType": "java-jar",
        "attachAllClusters": "true",
        "statuses": [
          {
            "clusterId": "0223-221237-tract",
            "status": "Attached",
            "message": "error message"
          },
        ]
    }
```

library *status* can be ["Attaching", "Attached", "Error", "Detaching", "Detached"]

Request:
POST with a library file
https://dbc_api_url/api/1.2/libraries/upload

data = {"libType": "python-egg", "name" : "mylib.egg", "folder": "/mylibraries", "attachToAllClusters": "false"}
files = {"uri": "./spark/python/test_support/userlib-0.1-py2.7.egg"}
Response:

```
{
    "id": "12345"
}
```

language can be ["java-jar", "python-pypi", "python-egg"]

Request:
POST with a python pypi package name
https://dbc_api_url/api/1.2/libraries/upload
data = {"libType": "python-pypi", "name" : "fuzzywuzzy", "folder": "/mylibraries", "attachToAllClusters": "false"}
files = {"uri": "fuzzywuzzy"}
Response:

```
{
    "id": "12345"
}
```

Request:
POST
https://dbc_api_url/api/1.2/libraries/delete
data = {"libraryId": "1234"}
Response:

```
{
    "id": "12345
}
```

Request:
POST
https://dbc_api_url/api/1.2/libraries/attach
data = {"libraryId": "1234", "clusterId" : "0223-221237-tract"}
Response:

```
{
    "id": "12345
}
```

clusterId can a specific cluster id, or *"__ALL_CLUSTERS".* The latter will schedule an attach action for the library to all clusters including new clusters.

Request:
POST
https://dbc_api_url/api/1.2/libraries/detach
data = {"libraryId": "1234", "clusterId" : "__ALL_CLUSTERS"}
Response:

```
{
    "id": "12345
}
```

clusterId can be a specific cluster id, or *"__ALL_CLUSTERS".* The latter will disable attaching to all clusters. Unless the library is specifically attached to the cluster, this will schedule a detaching action for that cluster.