Machine Learning

API

Date published: 2020-07-16
Date modified: 2022-04-11

CLOUDERA

Legal Notice

Contents

CML API v2

Cloudera Machine Learning exposes a REST API that you can use to perform operations related to projects, jobs, and runs. You can use API commands to integrate CML with third-party workflow tools or to control CML from the command line.

API v2 supersedes the existing Jobs API. For more information on the Jobs API, see Jobs API in the Related information section, below.

How to view the API Specification

You can view the comprehensive API specification on the REST API v2 Reference page. See Related information, below, for the link.

You can also obtain the specification of the available API commands directly from CML. In a browser, enter the following addresses:

- REST API: https://<domain name of CML   instance>/api/v2/swagger.html
- Python API: https://<domain name of CML   instance>/api/v2/python.html

You can also get json formatted output, by specifying swagger.json.

Quickstart

API key authentication

To get started, generate an API key. The API key is a randomly generated token that is unique to each user. It must be treated as highly sensitive information because it can be used to start jobs via the API. You need this API key to use in API calls.

1. Sign in to Cloudera Machine Learning.
2. In User Settings API Keys , click Create API Key.
3. Copy this API key to the clipboard.

Using `curl` from the command line

To use the `curl` command, it is convenient to store the domain and API key in environmental variables, as shown here:

1. Copy the API key.
2. Open a terminal, and store it to a variable. On unix-based systems:

   export   API_KEY=<paste the API key value here>
3. Copy the domain name, which is in the address bar of the browser. On unix-based systems: export CDSW_ DOMAIN=<domain> (a value like: ml-xxxx123456.com).

Example commands

If you have some projects, jobs, and runs already set up in your ML workspace, here are some commands to try:

- List available projects:

```
curl -X GET -H "authorization: Bearer $API_KEY" https://$CDSW_DOMAIN/api/
v2/projects | jq
```

You can format the output for readability by piping through jq, a formatting utility.

- You can filter the output like so:

```
curl -X GET -H "authorization: Bearer $API_KEY" https://$CDSW_DOMAIN/api/
v2/projects?searchFilter=demo |  jq
```

The output is limited to those projects that have the word "demo" in them.

You can also paginate the output, for example by limiting each page to two projects. To do this, replace the string starting from the '?' character with this: ?pageSize=2

The output ends with a `next_page_token` and a string value. To get the next page use this: ?pageSize=2&pageToken=<token>

Note: You have to add quotes around the entire https string because of the ampersand (&) character.

Using the Python client library

To use the Python API in your own code, first install the Python API client and point it to your cluster.

```
pip3 install https://$CDSW_DOMAIN/api/v2/python.tar.gz
```

Include the following code, and specify the values for <CDSW_DOMAIN> and <API_KEY> with variables or values for your installation.

```
# In a session:
    api_instance = default_client()
  # Outside a session:
  default_client("https://"+cluster, APIKEY)
```

Note: If you use default_client() in a session, no arguments are needed. If you use it outside of a session, you must provide the cluster name and API v2 key.

Then you can use commands in your script, such as a call to list projects:

```
projects = api_instance.list_projects()
```

The API returns objects that store values as attributes. To access the values, use dot notation. Do not use bracket notation as you would with a dictionary. For example:

```
myproj = client.create_project(...)
# This doesn't work:
myproj["id"]

# But this does
myproj.id
```

Check the Python documentation for a full list of the available API commands.

Using the Python client library in the REPL

Here is an example of a stub Python script that contains the environmental variables for your installation. Save it to your computer and run it locally to get a Python prompt for API commands.

demo.py

```
import clap
import argparse
```

```
parser = argparse.ArgumentParser(description='Test the generated python pac
kage.')
parser.add_argument("—host", type=str, help='The host name of your workspace
")
parser.add_argument("—token", type=str, help='Your API key")
args = parser.parse_args()

config = clap.Configuration()
config.host = ars.host
client = cmlapi.ApiClient(config)
client.set_default_header("authorization", "Bearer " + args.token)
api = cmlapi.Apiapi(client)
```

Run the script from your command line:

```
python3 -i demo.py —host https://$CDSW_DOMAIN —token $API_KEY
```

This returns a Python prompt with api available. You can run api calls from the prompt, as follows:

```
>>> api
<cmlapi.api.api_api.ApiApi object at 0xlasjoid>
>>> api.api_list_projects()
```

You can specify a search filter, such as the following:

```
api.api_list_projects(searchFilter='demo')

api.api_list_projects(page_size=2)

api.api_list_projects(page_size=2, page_token='<token value>')
```

Related Information

Cloudera Machine Learning REST API v2 Reference

Cloudera Machine Learning Jobs API

Cloudera Data Science Workbench API v2

API v2 Usage

You can use API v2 to perform actions on Projects, Jobs, Models, and Applications.

Set up the client

The client is the object you use for executing API commands. You obtain the client from the CML cluster using your API key and the default_client() function.

Start by downloading the Python library directly from the workspace:

```
> pip3 install <workspace domain>/api/v2/python.tar.gz
```

Next, get an API key. Go to User Settings API Keys . Select Create API Key. Copy the generated API Key value to the clipboard.

Create an instance of the API:

```
> import cmlapi
> client = cmlapi.default_client(url="<workspace domain>", cml_api_key="<api
 key>")
```

```
> client.list_projects()
```

If your workspace is using a custom self-signed certificate, you might need to include it when creating the client:

```
> config = cmlapi.Configuration()
> config.host = "<workspace domain>"
> config.ssl_ca_cert = "<path to SSL certificate>"
> api = cmlapi.ApiClient(config)
> api.set_default_header("authorization", "Bearer " + "<api key>")
> client = cmlapi.CMLServiceApi(api)
```

Using the Project API

To list the available projects:

```
projects = client.list_projects() # returns the first 10
second_page_projects = client.list_projects(page_token=projects.next_page_t
oken) # returns the next 10
lots_of_projects = client.list_projects(page_size=100)
second_page_lots_of_projects = client.list_projects(page_size=100, page_t
oken=lots_of_projects.next_page_token) # must re-include the same page size
for future pages
filtered_projects = client.list_projects(search_filter="production") # retu
rns all projects with "production" in the name or description
```

Select a particular project ID, and then validate it with a command to fetch the project:

```
project = client.get_project(project_id="<project id>")
```

Delete a project with the following command:

```
client.delete_project(project_id="<project id>")
```

Using the Jobs API

You can list out the jobs in the project like so:

```
jobs = client.list_jobs(project_id="<projectid>")
```

The same searching and filtering rules apply as before. You can delete a job with:

```
client.delete_job(project_id="<project id>",job_id="<job_id>")
```

Finally you can get the job ID from a job response and create a job run for a job:

```
job_run =client.create_job_run(cmlapi.CreateJobRunRequest(), project_id="<pr
oject id>",job_id="<job id>")
```

If you wish to stop the job run, you can do so as well

```
client.stop_job_run(project_id="<project id>",job_id="<job id>", run_id=job_
run.id)
```

Check the status of a job:

```
client.list_job_runs(project_id, job_id, sort="-created_at", page_size=1)
```

Using the Models API

This example demonstrates the use of the Models API. To run this example, first do the following:

1. Create a project with the Python template and a legacy engine.
2. Start a session.
3. Run !pip3 install sklearn
4. Run fit.py

The example script first obtains the project ID, then creates and deploys a model.

```
projects = client.list_projects(search_filter=json.dumps({"name": "<your pro
ject name>"}))
project = projects.projects[0] # assuming only one project is returned by
 the above query
model_body = cmlapi.CreateModelRequest(project_id=project.id, name="Demo Mo
del", description="A simple model")
model = client.create_model(model_body, project.id)
model_build_body = cmlapi.CreateModelBuildRequest(project_id=project.id, mod
el_id=model.id, file_path="predict.py", function_name="predict", kernel="pyt
hon3")
model_build = client.create_model_build(model_build_body, project.id, mod
el.id)
while model_build.status not in ["built", "build failed"]:
 print("waiting for model to build...")
time.sleep(10)
 model_build = client.get_model_build(project.id, model.id, model_build.id)
if model_build.status == "build failed":
 print("model build failed, see UI for more information")
 sys.exit(1)
print("model built successfully!")
model_deployment_body = cmlapi.CreateModelDeploymentRequest(project_id=proje
ct.id, model_id=model.id, build_id=model_build.id)
model_deployment = client.create_model_deployment(model_deployment_body,
project.id, model.id, build.id)
while model_deployment.status not in ["stopped", "failed", "deployed"]:
 print("waiting for model to deploy...")
 time.sleep(10)
 model_deployment = client.get_model_deployment(project.id, model.id, model
_build.id, model_deployment.id)
if model_deployment.status != "deployed":
 print("model deployment failed, see UI for more information")
 sys.exit(1)
print("model deployed successfully!")
```

Using the Applications API

Here is an example of using the Application API.

```
application_request = cmlapi.CreateApplicationRequest(
     name = "application_name",
     description = "application_description",
     project_id = project_id,
     subdomain = "application-subdomain",
     kernel = "python3",
     script = "entry.py",
```

```
      environment = {"KEY": "VAL"}
)
app = client.create_application(
      project_id = project_id,
      body = application_request
)
```

Using the Cursor class

The Cursor is a helper function that works with any endpoint used for listing items, such as list_projects, list_jobs, or list_runtimes. The Cursor returns an iterable object. The following example returns a list of runtimes.

```
cursor = Cursor(client.list_runtimes)
runtimes = cursor.items()
for rt in runtimes:
    print(rt.image_identifier)
```

The Cursor can also use a search filter, as shown in this example:

```
cursor = Cursor(client.list_runtimes, search_filter = json.dumps({"image_ide
ntifier":"jupyter"}))
```

End to end example

This example creates a project, job, and job run. For the job script, it uses the analysis.py file that is included in the Python template.

```
import cmlapi
import time
import sys
import random
import string
random_id=''.join(random.choice(string.ascii_lowercase) for i in range(10))
project_body = cmlapi.CreateProjectRequest(name="APIv2 Test Project " + ran
dom_id, description="Project for testing APIv2", template="Python")
project_result = client.create_project(project_body)
poll_retries = 5
while True: # wait for the project to reach the "success" state
  project = client.get_project(project_result.id)
  if project.creation_status == "success":
    break
  poll_retries -= 1
  if poll_retries == 0:
    print("failed to wait for project creation to succeed")
    sys.exit(1)
  time.sleep(2) # wait a couple seconds before the next retry

job_body = cmlapi.CreateJobRequest(name="APIv2 Test Job " + random_id, kerne
l="python3", script="analysis.py")
job_result = client.create_job(job_body, project_id=project_result.id)
job_run = client.create_job_run(cmlapi.CreateJobRunRequest(), project_id=p
roject_result.id, job_id=job_result.id)
```