IBM Analyics Engine Documentation

Release

Chris Snow

Mar 28, 2018

Contents:

This project is a Python library for working with the IBM Analytics Engine. The github repository for this library can be found here:

- https://github.com/snowch/ibm-analytics-engine-python

CHAPTER 1

Examples

This section shows example code snippets for working with this library.

1.1 Prerequisites

API Key

The lifecycle of an IBM Analytics Engine cluster is controlled through Cloud Foundry (e.g. create, delete, status operations). This python library requires an API key to work with the Cloud Foundry APIs. For more information on IBM Cloud API Keys including how to create and download an API Key, see https://console.bluemix.net/docs/iam/userid_keys.html#userapikey

Installation

Ensure you have installed this library.

Install with:

```
pip install --upgrade git+https://github.com/snowch/ibm-analytics-engine-python@master
```

1.2 Logging

Log level is controlled with the environment variable `LOG_LEVEL`.

You may set it programmatically in your code:

```
os.environ["LOG_LEVEL"] = "DEBUG"
```

Typical valid values are ERROR, WARNING, INFO, DEBUG. For a full list of values, see: https://docs.python.org/3/library/logging.html#logging-levels

1.3 Finding your space guid

Many operations in this library require you to specify a space guid. You can list the spaces guids for your account using this example:

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')
cf.print_orgs_and_spaces()
```

Alternatively, if you know your organisation name and space name, you can use the following:

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')

try:
    space_guid = cf.space_guid(org_name='your_org_name', space_name='your_space_name')
    print(space_guid)
except ValueError as e:
    # Space not found
    print(e)
```

1.4 Create Cluster

This example shows how to create a basic spark cluster.

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI
from ibm_analytics_engine import IAE, IAEServicePlanGuid

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')

space_guid = cf.space_guid(org_name='your_org_name', space_name='your_space_name')

iae = IAE(cf_client=cf)

cluster_instance_guid = iae.create_cluster(
    service_instance_name='SPARK_CLUSTER',
    service_plan_guid=IAEServicePlanGuid.LITE,
    space_guid=space_guid,
    cluster_creation_parameters={
        "hardware_config": "default",
        "num_compute_nodes": 1,
        "software_package": "ae-1.0-spark",
    }
)
print('>> IAE cluster instance id: {}'.format(cluster_instance_guid))

# This call blocks for several minutes.  See the Get Cluster Status example
# for alternative options.

status = iae.status(
    cluster_instance_guid=cluster_instance_guid,
    poll_while_in_progress=True)
```

```
print('>> Cluster status: {}'.format(status))
```

The above example creates a LITE cluster. See *IBMServicePlanGuid* for the available service plan guids.

1.5 Delete Cluster

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI, CloudFoundryException
from ibm_analytics_engine import IAE

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')

iae = IAE(cf_client=cf)
try:
    iae.delete_cluster(
        cluster_instance_guid='12345-12345-12345-12345',
        recursive=True)

    print('Cluster deleted.')
except CloudFoundryException as e:
    print('Unable to delete cluster: ' + str(e))
```

1.6 Get or Create Credentials

```
import json
from ibm_analytics_engine.cf.client import CloudFoundryAPI
from ibm_analytics_engine import IAE

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')
iae = IAE(cf_client=cf)

vcap_json = iae.get_or_create_credentials(cluster_instance_guid='12345-12345-12345-
↪12345')

# prettify json
vcap_formatted = json.dumps(vcap_json, indent=4, separators=(',', ': '))

print(vcap_formatted)
```

To save the returned data to disk, you can do something like:

```
with open('./vcap.json', 'w') as vcap_file:
    vcap_file.write(vcap_formatted)
```

1.7 Get Cluster Status

To return the Cloud Foundry status:

```
import time
from ibm_analytics_engine.cf.client import CloudFoundryAPI
from ibm_analytics_engine import IAE

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')
iae = IAE(cf_client=cf)

while True:
    status = iae.status(cluster_instance_guid='12345-12345-12345-12345')
    if status == 'succeeded' or status == 'failed': break
    time.sleep(60)

print(status)
```

Alternative option to poll for the Cloud Foundry status. Note that this approach can block for many minutes while a cluster is being provisioned. While it is blocked, there is no progress output:

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI
from ibm_analytics_engine import IAE

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')
iae = IAE(cf_client=cf)

status = iae.status(
    cluster_instance_guid='12345-12345-12345-12345',
    poll_while_in_progress=True)

print(status)
```

To return the Data Platform API status:

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI
from ibm_analytics_engine import IAE

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')

iae = IAE(cf_client=cf)

vcap = iae.get_or_create_credentials(cluster_instance_guid='12345-12345-12345-12345')

status = iae.dataplatform_status(vcap)

print(status)
```

1.8 List Clusters

```
from ibm_analytics_engine.cf.client import CloudFoundryAPI
from ibm_analytics_engine import IAE

cf = CloudFoundryAPI(api_key_filename='your_api_key_filename')

space_guid = cf.space_guid(org_name='your_org_name', space_name='your_space_name')

iae = IAE(cf_client=cf)
```

```
for i in iae.clusters(space_guid=space_guid):
    print(i)
```

1.9 Jupyter Notebook Gateway

This is an example script for running a docker notebook that connects to the cluster using the JNBG protocol and the credentials in your vcap.json file.

```
#!/bin/bash

export VCAP_STR="$(cat vcap.json)"

KG_HTTP_USER=$(python -c "import json, os; print(json.loads(os.environ['VCAP_STR'])[
↪'cluster']['user'])")
KG_HTTP_PASS=$(python -c "import json, os; print(json.loads(os.environ['VCAP_STR'])[
↪'cluster']['password'])")
KG_HTTP_URL=$(python -c "import json, os; print(json.loads(os.environ['VCAP_STR'])[
↪'cluster']['service_endpoints']['notebook_gateway'])")
KG_WS_URL=$(python -c "import json, os; print(json.loads(os.environ['VCAP_STR'])[
↪'cluster']['service_endpoints']['notebook_gateway_websocket'])")

# Create a directory for the notebooks so they don't disappear when the docker␣
↪constainer shuts down
if [ ! -d notebooks ]
then
   mkdir notebooks
fi

docker run -it --rm \
        -v $(pwd)/notebooks:/tmp/notebooks \
        -e KG_HTTP_USER=$KG_HTTP_USER \
        -e KG_HTTP_PASS=$KG_HTTP_PASS \
        -e KG_URL=$KG_HTTP_URL \
        -e KG_WS_URL=$KG_WS_URL \
        -p 8888:8888 \
        biginsights/jupyter-nb-nb2kg

# Open a browser window to: http://127.0.0.1:8888
```

CHAPTER 2

API Docs

A python library for working with IBM Analytics Engine

2.1 Service Instance Operations

Lifecycle operations for an IAE service instance.

class ibm_analytics_engine.**IAE**(*cf_client*)

This class provides methods for working with IBM Analytics Engine (IAE) deployment operations. Many of the methods in this calls are performed by calling the Cloud Foundry Rest API (https://apidocs.cloudfoundry.org/272/). The Cloud Foundry API is quite abstract, so this class provides methods names that are more meaningful for those just wanting to work with IAE.

This class does not save the state from the Cloud Foundry operations - it retrieve all state from Cloud Foundry as required.

__init__(*cf_client*)

Create a new instance of the IAE client.

Parameters **cf_client** (CloudFoundryAPI) – The object that makes the Cloud Foundry rest API calls.

clusters(*space_guid*, *short=True*, *status=None*)

Returns a list of clusters in the *space_guid*

Parameters

- **space_guid** (str) – The space_guid to query for IAE clusters.
- **short** (bool, optional) – Whether to return short (brief) output. If false, returns the full Cloud Foundry API output.
- **status** (str, optional) – Filter the return only the provided status values.

Returns

If the *short=True*, this method returns: *[(cluster_name, cluster_guid, last_operation_state), ...]*

The *last_operation_status* may be:

- *in progress*
- *succeeded*
- *failed*

Return type `list`

create_cluster(*service_instance_name*, *service_plan_guid*, *space_guid*, *cluster_creation_parameters*)

Create a new IAE Cluster‘

Parameters

- **service_instance_name** (`str`) – The name you would like for the Cluster.
- **service_plan_guid** (*IAEServicePlanGuid*) – The guid representing the type of Cluster to create.
- **space_guid** (`str`) – The space guid where the Cluster will be created.
- **cluster_creation_parameters** (`dict`) – The cluster creation parameters. An example cluster creation parameters is shown below:

```
{
    “num_compute_nodes”: 1,
    ”hardware_config”: “Standard”,
    ”software_package”: “ae-1.0-spark”,
    ”customization”: [{
        “name”: “action1”,
        ”type”: “bootstrap”,
        ”script”: {
            “source_type”: “http”,
            ”script_path”: “http://path/to/your/script”
            },
        ”script_params”: []
        }]
}
```

Returns The cluster_instance_guid

Return type `str`

More api docs coming soon ...

2.2 Spark Livy Operations

Not implemented yet.

2.3 Spark SSH Operations

Not implemented yet.

2.4 Ambari Operations

Not implemented yet.

2.5 Cloud Foundry Operations

class ibm_analytics_engine.**CloudFoundryAPI**(*api_key=None*, *api_key_filename=None*, *api_endpoint='https://api.ng.bluemix.net'*, *provision_poll_timeout_mins=30*)

2.6 Utility Classes

2.6.1 IBMServicePlanGuid

class ibm_analytics_engine.**IAEServicePlanGuid**

Service Plan Guid for IBM Analytics Engine.

LITE = 'acb06a56-fab1-4cb1-a178-c811bc676164'

IBM Analytics Engine 'Lite' plan.

STD_HOURLY = '9ba7e645-fce1-46ad-90dc-12655bc45f9e'

IBM Analytics Engine 'Standard Hourly' plan.

STD_MONTHLY = 'f801e166-2c73-4189-8ebb-ef7c1b586709'

IBM Analytics Engine 'Standard Monthly' plan.

CHAPTER 3

Example notebook - Create Cluster

This example uses a python library for working with an IAE instance.

IBM Analytics Engine Python Library links:

- documentation is on readthedocs
- source code repository is on github
- this notebook is on github

```
In [ ]: ! pip install --quiet --upgrade git+https://github.com/snowch/ibm-analytics-engine-python@mas
```

```
In [ ]: from ibm_analytics_engine import CloudFoundryAPI, CloudFoundryAPI
        from ibm_analytics_engine import IAE, IAEServicePlanGuid, IAEClusterSpecificationExamples
```

We use an IBM Cloud API key to work with an IAE Instance. You can create an API using the bluemix CLI tools, e.g.

```
bluemix iam api-key-create My_IAE_Key -d "This is my IAE API key" -f my_api_key.json
```

Alternatively, follow these instructions to create an API key using the IBM Cloud web console and then save it in a secure location.

```
In [ ]: cf = CloudFoundryAPI(api_key_filename='./my_api_key.json')
```

You aren't restricted to just using an API key file. If you have the API key value, you can do this:

```
from getpass import getpass
api_key = getpass("Enter your api key: ")

cf = CloudFoundryAPI(api_key=api_key)
```

```
In [ ]: # Provide your organizaton name and space name:

        SPACE_GUID = cf.space_guid(org_name='my_org_name', space_name='my_space_name')
        print(SPACE_GUID)
```

If you couldn't find your space guid, try printing out all your orgs and spaces:

```
cf.print_orgs_and_spaces()
```

```
In [ ]: # We interact with the IBM Analytics Engine through the IAE class.
        # Let's create an instance of it:

        iae = IAE(cf_client=cf)
```

```
In [ ]: # List the clusters in the space

        iae.clusters(space_guid=SPACE_GUID)
```

```
In [ ]: cluster_guid = iae.create_cluster(service_instance_name = 'MY_SPARK_CLUSTER',
                            service_plan_guid = IAEServicePlanGuid.LITE,
                            cluster_creation_parameters = {
                                 "hardware_config": "default",
                                 "num_compute_nodes": 1,
                                 "software_package": "ae-1.0-spark",
                             },
                            space_guid = SPACE_GUID)
```

Alternative options for service_plan_guid:

- `IAEServicePlanGuid.STD_HOURLY`
- `IAEServicePlanGuid.STD_MONTHLY`

There are also some examples of cluster_creation_paramters in `IAEClusterSpecificationExamples` class:

```
IAEClusterSpecificationExamples.SINGLE_NODE_BASIC_SPARK = {
    'num_compute_nodes': 1,
    'hardware_config': 'default',
    'software_package': 'ae-1.0-spark'
    }
```

and:

```
IAEClusterSpecificationExamples.SINGLE_NODE_BASIC_HADOOP = {
    'num_compute_nodes': 1,
    'hardware_config': 'default',
    'software_package': 'ae-1.0-hadoop-spark'
    }
```

These have been provided so you don't have to remember the parameters for creating a default basic cluster.

You would use them like this:

```
iae.create_cluster(...,
    cluster_creation_parameters = IAEClusterSpecificationExamples.SINGLE_NODE_BASIC_
↪SPARK,
    ...)
```

```
In [ ]: # Poll the cluster until provisioning has finished

        import time
        while True:
            status = iae.status(cluster_instance_guid=cluster_guid)
            print(status)
            if status == 'succeeded' or status == 'failed': break
            time.sleep(60)
```

In [ ]:
```
# Only run this cell after the previous cell has finished with the status 'succeeded',
# otherwise you will receive an error trying to get or create the credentials.

import json

# get the credentials data for the cluster in vcap json format
vcap = iae.get_or_create_credentials(cluster_instance_guid=cluster_guid)

# print the credentials out
vcap_formatted = json.dumps(vcap, indent=4, separators=(',', ': '))
print(vcap_formatted)

# save the credentials to a file
with open('./vcap.json', 'w') as vcap_file:
    vcap_file.write(vcap_formatted)
```

In [ ]:
```
# Grab the ambari console url

print(vcap['cluster']['service_endpoints']['ambari_console'])
```

In [ ]:
```
# Delete the cluster.  Recursive=True will delete service bindings, service keys,
# and routes associated with the service instance.

iae.delete_cluster(cluster_guid, recursive=True)
```

CHAPTER 4

Example notebook - nb2kg

This notebook is an example of connecting a jupyter notebook to IBM Analytics Engine (IAE) using nb2kg. It is expected that you will run this notebook on a local jupyter environment (i.e. not from DSX).

The notebook connecting to IAE is spun up in a docker container. This is because:

- When you enable the nb2kg extension, all kernels run on the configured Kernel Gateway, instead of on the Notebook server host. The extension does not support local kernels. If nb2kg is installed in your local notebook environment, it would prevent you from running local kernels. Thus using docker prevents me from corrupting your local notebook environment.
- Setting up a notebook environment with all the dependencies for nb2kg can be tricky. Docker allows me to provide you with an environment that is pre-configured.

Python is used to interact with docker because this was easier to script in a notebook. However, if you run docker with `sudo`, this notebook may not work for you.

WARNING: This project is just a demo of connecting to IAE using nb2kg. Your mileage may vary.

```
In [ ]: ! pip install --quiet docker
```

Load the IBM Analytics Engine (IAE) credentials. See *this example notebook* for creating an IAE instance and saving the vcap.json credentials file.

```
In [ ]: import json
        vcap = json.load(open('./vcap.json'))
```

```
In [ ]: import docker
        client = docker.from_env()
        api_client = docker.APIClient()
```

The docker images can be quite large and take a long time to load. Here we load the parent image and regularly print some output so you can see what is going on.

```
In [ ]: import json

        lines_processed = 0

        api_client = docker.APIClient()
        for line in api_client.pull('jupyter/minimal-notebook:fa77fe99579b', tag=None, stream=True):
```

```
            if lines_processed % 25 == 0:
                try:
                    status = json.loads(str(line)[2:-5]) # strip quotes and newline
                    if 'progressDetail' in status:
                        print(status['progressDetail'])
                except:
                    pass
            lines_processed += 1
```

I have created a custom docker environment that uses nb2kg. This environment has been kept as simple as possible to make it easy for you to adapt to to your own requirements.

```
In [ ]: image = client.images.build(
            path='https://github.com/snowch/docker_jupyter_notebook_kg.git',
            tag='docker_jupyter_notebook_kg')

        print(image.tags)
```

Delete any containers hanging around from a previous run of this notebook.

WARNING: Ensure you backup any work you want to keep before running this command.

```
In [ ]: from datetime import datetime as dt
        import dateutil.parser

        for cont in client.containers.list(filters={ 'name': 'iae_nb2kg_example' }):
            created = dt.utcnow() - dateutil.parser.parse(cont.attrs['Created']).replace(tzinfo=None)
            print("Name: {} | Status: {} | Age (H:M:S): {}".format(
                                cont.attrs['Name'],
                                cont.attrs['State']['Status'],
                                created
                                ))
            #print(json.dumps(cont.attrs, indent=4, sort_keys=True))
            cont.kill()
```

Run the notebook. You may need to change the LOCAL_PORT if this port is not free on your local machine.

Change the LOCAL_NOTEBOOKS_FOLDER to a folder on your local machine where you want your notebooks in the 'work' folder to be saved.

```
In [ ]: LOCAL_PORT = 8899
        LOCAL_NOTEBOOKS_FOLDER = '/Users/snowch/Desktop/notebooks'

        container = client.containers.run(
            image = image,
            volumes = { LOCAL_NOTEBOOKS_FOLDER : '/home/jovyan/work' },
            ports = {str(LOCAL_PORT)+'/tcp': LOCAL_PORT},
            environment = {
                'NB_PORT': LOCAL_PORT,
                'KG_HTTP_USER': vcap['cluster']['user'],
                'KG_HTTP_PASS': vcap['cluster']['password'],
                'KG_URL': vcap['cluster']['service_endpoints']['notebook_gateway'],
                'KG_WS_URL': vcap['cluster']['service_endpoints']['notebook_gateway_websocket'],
                'KG_CONNECT_TIMEOUT': '50.0',
                'KG_REQUEST_TIMEOUT': '40.0'
            },
            detach=True,
            stdout=True,
            stderr=True,
            remove=True,
            name='iae_nb2kg_example'
            )
```

The next cell prints the log output. Ensure there are no errors reported.

You should see a url, e.g.

```
Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://0.0.0.0:8899/?token=12345
```

Click on the url in the output to open it in your browser.

From here, you should be able to create a new notebook:

```
In [ ]: from IPython.lib import backgroundjobs as bg
        jobs = bg.BackgroundJobManager()

        def printlogs():
            for line in container.logs(stream=True):
                print(str(line)[2:-3]) # strip quotes and newline
            sys.stdout.flush()

        jobs.new('printlogs()')
        jobs.status()
```

finally, remove the notebook docker container

```
In [ ]: container.kill()
```

Install with:

```
pip install --upgrade git+https://github.com/snowch/ibm-analytics-engine-python@master
```

NOTE: This documentation is a work-in-progress. It will be complete within the next few days/weeks. Please come back soon . . .

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

Symbols

C

I

L

S