LogicMonitor DATA SDK Python

Release 0.0.1.beta1

Logicmonitor

Feb 10, 2021

CONTENTS:

This Python Library for ingesting the metrics into the LogicMonitor Platform

CHAPTER

ONE

PUSHMETRICS - METRICS INGESTION

1.1 Overview

LogicMonitor's Push Metrics feature allows you to send metrics directly to the LogicMonitor platform via a dedicated API, removing the need to route the data through a LogicMonitor Collector. Once ingested, these metrics are presented alongside all other metrics gathered via LogicMonitor, providing a single pane of glass for metric monitoring and alerting.

More details are available on support site

The `logicmonitor_data_sdk` module provides

- *`logicmonitor_data_sdk.api.metrics`*: a HTTP Api client for ingesting the metrics data.

1.2 Requirements.

Python 2.7 and 3.4+

1.3 Installation

1.3.1 pip install

Install from PyPI.

```
pip install logicmonitor_data_sdk
```

Then import the package:

```
import logicmonitor_data_sdk
```

1.4 Getting Started

Please follow the *Installation* and then run below a working example for submitting the disk metrics to your LM account. This script will monitor the Usage, Free and Total of the disk at every 5 sec interval.

```
import logging
import os
import time

import psutil as psutil

import logicmonitor_data_sdk
from logicmonitor_data_sdk.api.metrics import Metrics
from logicmonitor_data_sdk.api.response_interface import ResonseInterface
from logicmonitor_data_sdk.models import Resource, DataSource, DataPoint, \
  DataSourceInstance

logger = logging.getLogger('lmdata.api')
logger.setLevel(logging.INFO)

configuration = logicmonitor_data_sdk.Configuration()
# For debug log, set the value to True
configuration.debug = False


class MyResponse(ResonseInterface):
  """
  Sample callback to handle the response from the REST endpoints
  """

  def success_callback(self, request, response, status, request_id):
    logger.info("%s: %s: %s", response, status, request_id)

  def error_callback(self, request, response, status, request_id, reason):
    logger.error("%s: %s: %s %s", response, status, reason, request_id)


def MetricRequest():
  """
  Main function to get the CPU values using `psutil` and send to Metrics REST endpoint
  """
  device_name = os.uname()[1]
  resource = Resource(ids={'system.displayname': device_name}, name=device_name,
                      create=True)
  datasource = DataSource(name="DiskUsingSDK")
  datapoints = ['total', 'used', 'free']
  metric_api = Metrics(batch=True, interval=30, response_callback=MyResponse())
  while True:
    partitions = psutil.disk_partitions()
    for p in partitions:
      # Using the device as instance name. We can use the mountpoint as well.

      instance_name = p.device
      usage = psutil.disk_usage(instance_name)._asdict()

      # Create the instance object for every device. Name should not have the
      # special characters so replacing it with the '-'.
```

(continues on next page)

(continued from previous page)

```
      instance = DataSourceInstance(name=instance_name.replace('/', '-'),
                                    display_name=instance_name)
      for one_datapoint in datapoints:
        datapoint = DataPoint(name=one_datapoint)
        values = {str(int(time.time())): str(usage[one_datapoint])}
        metric_api.send_metrics(resource=resource,
                                datasource=datasource,
                                instance=instance,
                                datapoint=datapoint,
                                values=values)
    time.sleep(5)


if __name__ == "__main__":
  MetricRequest()
```

Then run the program as:

```
pip install psutil
LM_COMPANY=<ACOUNT_NAME> LM_ACCESS_ID=<ID> LM_ACCESS_KEY='<KEY>' python disk_metrics.
↪py
```

1.5 Configuration

SDK must be configured with logicmonitor_data_sdk.Configuration(). The account name, an API key and its id are required.

class logicmonitor_data_sdk.configuration.**Configuration**(***kwargs*)

This model is used to defining the configuration.

Parameters

- **company** (str) – The account name. If it is not provided then we will use the 'LM_COMPANY' environment variable.
- **authentication** (dict of *id* and *key*) – LogicMonitor supports verious types of the authentication. This variable will be used to specify the authentication key. If it is not provided then 'LM_ACCESS_ID' and 'LM_ACCESS_KEY' environment variable will be used to find the id and key.

Examples

```
>>> import logicmonitor_data_sdk
>>> conf = logicmonitor_data_sdk.Configuration(company="ACCOUNT_NAME",␣
↪authentication={'id': 'API_ACCESS_ID', 'key': 'API_ACCESS_KEY', 'type' : 'LMv1'}
↪)
```

---

property async_req

The async request.

Parameters **value** – enable async request string.

Type bool

property debug
Debug status

Parameters **value** – The debug status, True or False.

Type bool

property logger_file
The logger file.

If the logger_file is None, then add stream handler and remove file handler. Otherwise, add file handler and remove stream handler.

Parameters **value** – The logger_file path.

Type str

property logger_format
The logger format.

The logger_formatter will be updated when sets logger_format.

Parameters **value** – The format string.

Type str

to_debug_report()
Gets the essential information for debugging.

Returns The report for debugging.

1.6 API Calls

All URIs are relative to *https://<account_name>.logicmonitor.com/rest*

1.6.1 Usage

Be sure to initialize the client using *Configuration* and then use *Metrics Ingestion API*.

1.6.2 Metrics Ingestion API

Metrics API client: It formats and submit REST API calls to LogicMonitor.

class logicmonitor_data_sdk.api.metrics.**Metrics**(*batch=True*, *interval=30*, *response_callback=None*, *api_client=None*)
This API client is for ingesting the metrics in LogicMonitor and updating the properties of the resource or instance.

Parameters

- **batch** (bool) – Enable the batching support.
- **interval** (int) – Batching flush interval. If batching is enabled then after that second we will flush the data to REST endpoint.
- **response_callback** (*logicmonitor_data_sdk.api.response_interface.ResonseInterface*) – Callback for response handling.

- **api_client** (logicmonitor_data_sdk.api_client.ApiClient) – The RAW HTTP REST client.

Examples

```
>>> from logicmonitor_data_sdk.api.metrics import Metrics
>>> from logicmonitor_data_sdk.configuration import Configuration
>>> conf = Configuration(company="ACCOUNT_NAME",   authentication={'id': 'API_
↪ACCESS_ID', 'key': 'API_ACCESS_KEY', 'type' : 'LMv1'})
>>> # Create the Metrics client with batching support and flush interval as 30␣
↪sec.
>>> metricsApi = Metrics(batch=True, interval=30)
```

send_metrics(***kwargs*)

This send_metrics method is used to send the metrics to rest endpoint.

Parameters

- **resource** (*logicmonitor_data_sdk.models.resource.Resource*) – The Resource object.
- **datasource** (*logicmonitor_data_sdk.models.datasource.DataSource*) – The datasource object.
- **instance** (*logicmonitor_data_sdk.models.datasource_instance.DataSourceInstance*) – The instance object.
- **datapoint** (*logicmonitor_data_sdk.models.datapoint.DataPoint*) – The datapoint object.
- **values** (dict) – The values dictionary.

Returns If in *Metrics* batching is enabled then None Otherwise the REST response will be return.

Examples

```
>>> import time
>>> from logicmonitor_data_sdk.api.metrics import Metrics
>>> from logicmonitor_data_sdk.configuration import Configuration
>>> from logicmonitor_data_sdk.models.resource import Resource
>>> from logicmonitor_data_sdk.models.datasource import DataSource
>>> from logicmonitor_data_sdk.models.datasource_instance import␣
↪DataSourceInstance
>>> from logicmonitor_data_sdk.models.datapoint import DataPoint
>>>
>>> conf = Configuration(company="ACCOUNT_NAME", authentication={'id': 'API_
↪ACCESS_ID', 'key': 'API_ACCESS_KEY', 'type' : 'LMv1'})
>>> # Create the Metrics client with batching disabled
>>> metric_api = Metrics(batch=False)
>>> # Create the Resource object using the 'system.deviceId' properties.
>>> resource = Resource(ids={"system.hostname": "SampleDevice"}, create=True,␣
↪name="SampleDevice", properties={'using.sdk': 'true'})
>>> # Create the LMDataSource object for CPU monitoring
>>> ds = DataSource(name="CPU")
>>> # Create the DataSourceInstance object for CPU-0 instance monitoring
>>> instance = DataSourceInstance(name="CPU-0")
```

(continues on next page)

(continued from previous page)

```
>>> # Create the DataPoint object for cpu-time
>>> dp = DataPoint(name='cpu_time', aggregation_type='sum')
>>> metric_api.send_metrics(resource=resource, datasource=ds,␣
↪instance=instance, datapoint=dp, values={ time.time() : '23'})
```

1.7 Models

1.7.1 Resource

class logicmonitor_data_sdk.models.resource.**Resource**(*ids*, *name*, *description=None*, *properties=None*, *create=False*)

This model is used to define the resource.

Parameters

- **ids** (dict) – An array of existing resource properties that will be used to identify the resource. See Managing Resources that Ingest Push Metrics for information on the types of properties that can be used. If no resource is matched and the create parameter is set to TRUE, a new resource is created with these specified resource IDs set on it. If the system.displayname and/or system.hostname property is included as resource IDs, they will be used as host name and display name respectively in the resulting resource.
- **name** (str) – Resource unique name. Only considered when creating a new resource.
- **properties** (dict of str, optional) – New properties for resource. Updates to existing resource properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.
- **description** (str, optional) – Resource description. Only considered when creating a new resource.
- **create** (bool, optional) – Do you want to create the resource.

Examples

```
>>> from logicmonitor_data_sdk.models.resource import Resource
>>> # Create the Resource object using the 'system.deviceId' properties.
>>> resource = Resource(ids={'system.deviceId' : '1234'}, name='DeviceName',␣
↪create=False)
```

property create

Gets the create flag.

Returns create flag.

Return type bool

property description

Resource description. Only considered when creating a new resource.

Returns The description of this Resource.

Return type str

property ids
An array of existing resource properties that will be used to identify the resource. See Managing Resources that Ingest Push Metrics for information on the types of properties that can be used. If no resource is matched and the create parameter is set to TRUE, a new resource is created with these specified resource IDs set on it. If the system.displayname and/or system.hostname property is included as resource IDs, they will be used as host name and display name respectively in the resulting resource.

Returns The ids of this Resource.

Return type dict

property name
Resource unique name. Only considered when creating a new resource.

Returns The name of this Resource.

Return type str

property properties
New properties for resource. Updates to existing resource properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.

Returns The properties of this Resource.

Return type dict

1.7.2 DataSource

class logicmonitor_data_sdk.models.datasource.**DataSource**(*name*, *display_name=None*, *group=None*, *id=None*)

This model is used to defining the datasource object.

Parameters

- **name** (str) – DataSource unique name. Used to match an existing DataSource. If no existing DataSource matches the name provided here, a new DataSource is created with this name.
- **display_name** (str, optional) – DataSource display name. Only considered when creating a new DataSource.
- **group** (str, optional) – DataSource group name. Only considered when DataSource does not already belong to a group. Used to organize the DataSource within a DataSource group. If no existing DataSource group matches, a new group is created with this name and the DataSource is organized under the new group.
- **id** (int, optional) – DataSource unique ID. Used only to match an existing DataSource. If no existing DataSource matches the provided ID, an error results.

Examples

```
>>> from logicmonitor_data_sdk.models.datasource import DataSource
>>> # Create the DataSource object for CPU monitoring
>>> ds = DataSource(name='CPU')
```

property display_name
DataSource display name. Only considered when creating a new DataSource.

Returns The display_name of this DataSource.

Return type str

property group
DataSource group name. Only considered when DataSource does not already belong to a group. Used to organize the DataSource within a DataSource group. If no existing DataSource group matches, a new group is created with this name and the DataSource is organized under the new group.

Returns The group of this DataSource.

Return type str

property id
DataSource unique ID. Used only to match an existing DataSource. If no existing DataSource matches the provided ID, an error results.

Returns The id of this DataSource. # noqa: E501

Return type int

property name
DataSource unique name. Used to match an existing DataSource. If no existing DataSource matches the name provided here, a new DataSource is created with this name.

Returns The data_source of this DataSource.

Return type str

1.7.3 DataSourceInstance

class logicmonitor_data_sdk.models.datasource_instance.**DataSourceInstance**(*name*, *description=None*, *display_name=None*, *properties=None*)

This model is used to defining the datasource object.

Parameters

- **name** (str) – Instance name. If no existing instance matches, a new instance is created with this name.
- **display_name** (str, optional) – Instance display name. Only considered when creating a new instance.
- **properties** (dict of str, optional) – New properties for instance. Updates to existing instance properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.

Examples

```
>>> from logicmonitor_data_sdk.models.datasource_instance import␣
↪DataSourceInstance
>>> # Create the DataSourceInstance object for CPU-0 instance monitoring
>>> instance = DataSourceInstance(name='CPU-0')
```

property display_name
Instance display name. Only considered when creating a new instance.

Parameters **display_name** – The display_name of this DataSourceInstance.

Type str

property name
Instance name. If no existing instance matches, a new instance is created with this name.

Returns The name of this DataSourceInstance.

Return type str

property properties
New properties for instance. Updates to existing instance properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.

Returns The properties of this DataSourceInstance.

Return type MapStringString

1.7.4 DataPoint

class logicmonitor_data_sdk.models.datapoint.**DataPoint**(*name*, *aggregation_type=None*, *description=None*, *type=None*)

This model is used to defining the datapoint object.

Parameters

- **name** (str) – Datapoint name. If no existing datapoint matches for specified DataSource, a new datapoint is created with this name.
- **aggregation_type** (str, optional) – The aggregation method, if any, that should be used if data is pushed in sub-minute intervals. Only considered when creating a new datapoint. See the About the Push Metrics REST API section of this guide for more information on datapoint value aggregation intervals.
- **description** (str, optional) – Datapoint description. Only considered when creating a new datapoint.
- **type** (str, optional) – Metric type as a number in string format. Only considered when creating a new datapoint.

Examples

```
>>> from logicmonitor_data_sdk.models.datapoint import DataPoint
>>> # Create the DataPoint object for cpu_time
>>> dp = DataPoint(name='cpu_time', aggregation_type='sum')
```

property aggregation_type
The aggregation method, if any, that should be used if data is pushed in sub-minute intervals. Only considered when creating a new datapoint.

Returns The type of this DataPoint.

Return type str

property description
Datapoint description. Only considered when creating a new datapoint.

Returns The description of this DataPoint.

Return type str

property name
Datapoint name. If no existing datapoint matches for specified DataSource, a new datapoint is created with this name.

Returns The name of this DataPoint.

Return type str

property type
Metric type as a number in string format. Only considered when creating a new datapoint.

Returns The aggregation_type of this DataPoint.

Return type str

1.7.5 ResonseInterface

class logicmonitor_data_sdk.api.response_interface.**ResonseInterface**
This is the callback interface for handling the response. End user can create his own class using this one to get the response status.

classmethod error_callback(*request*, *response*, *status*, *request_id*, *reason*)
This callback gets invoked for any error or exception from the end REST endpoint.

Parameters

- **request** (dict of str) – The json payload send to REST endpoint.
- **response** (dict of str) – Response received from the REST endpoint.
- **status** (int) – HTTP status code.
- **request_id** (str) – Unique request id generated by Rest endpoint.
- **reason** (str) – The reason for error.

classmethod success_callback(*request*, *response*, *status*, *request_id*)
This callback gets invoked for successful response from the end REST endpoint.

Parameters

- **request** (dict of str) – The json payload send to REST endpoint.
- **response** (dict of str) – Response received from the REST endpoint.
- **status** (int) – HTTP status code.
- **request_id** (str) – Unique request id generated by Rest endpoint.

1.7.6 Get in Touch

If you have questions in general, reach out to our support@logicmonitor.com

PYTHON MODULE INDEX

l

INDEX

N

P

R

S

T