Py Healthchecks.Io

Andrew Herrington

May 14, 2024

CONTENTS

1 Features 3

2 Requirements 5

3 Installation 7

4 Usage 9

5 Contributing 11

6 License 13

7 Issues 15

8 Credits 17

Python Module Index 43

Index 45

A python client for healthchecks.io. Supports the management api and ping api.

CHAPTER ONE

FEATURES

- Sync and Async clients based on HTTPX
- Supports the management api and the ping api
- Supports Healthchecks.io SAAS and self-hosted instances

CHAPTER TWO

REQUIREMENTS

- httpx
- pytz
- pydantic

CHAPTER THREE

INSTALLATION

You can install *Py Healthchecks.Io* via pip from PyPI:

```
$ pip install healthchecks-io
```

CHAPTER

FOUR

USAGE

Please see the Usage for details.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the Contributor Guide.

CHAPTER

SIX

LICENSE

Distributed under the terms of the MIT license, *Py Healthchecks.Io* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Usage

This package implements the Healthchecks.io Management and Ping APIs as documented here https://healthchecks.io/docs/api/.

8.1.1 Context Manager

Either the Client or AsyncClient can be used as a ContextManager (or Async Context Manager)

```python
from healthchecks_io import Client, CheckCreate

with Client(api_key="myapikey") as client:
    check = client.create_check(CheckCreate(name="New Check", tags="tag1 tag2"))
print(check)
```

This is probably the easiest way to use the Clients for one-off scripts. If you do not need to keep a client open for multiple requests, just use the context manager.

Note: When using either of the client types as a context manager, the httpx client underlying the client will be closed when the context manager exits.

Since we allow you to pass in a client on creation, its possible to use a shared client with this library. If you then use the client as a contextmanager, it will close that shared client.

Just a thing to be aware of!

8.1.2 Sync

Instantiate a Client

If you want to work with the healthchecks.io API for the SaaS healthchecks, you can create a client like below:

```
from healthchecks_io import Client

client = Client(api_key="myapikey", ping_key="optional_ping_key")
```

If you are using a self-hosted healthchecks instance, you can set the api url when creating the client.

```
from healthchecks_io import Client

client = Client(api_key="myapikey",
                api_url="https://myhealthchecksinstance.com/api/",
                ping_key="optional_ping_key")
```

Creating a new Check

```
from healthchecks_io import Client, CheckCreate

client = Client(api_key="myapikey")

check = client.create_check(CheckCreate(name="New Check", tags="tag1 tag2"))
print(check)
```

Getting a Check

```
from healthchecks_io import Client

client = Client(api_key="myapikey")

check = client.get_check(check_id="mychecksuuid")
print(check)
```

Pinging a Check

```
from healthchecks_io import Client

client = Client(api_key="myapikey")
result, text = client.success_ping(uuid="mychecksuuid")
print(text)
```

8.1.3 Async

If you want to use the client in an async program, use AsyncClient instead of Client

```python
import asyncio
from healthchecks_io import AsyncClient, CheckCreate

async def main():
    client = AsyncClient(api_key="myapikey")

    check = await client.create_check(CheckCreate(name="New Check", tags="tag1 tag2"))
    print(check)

if __name__ == "__main__":
    asyncio.run(main())
```

8.1.4 CheckTrap

Ever wanted to run some code and wrape it in a healthcheck check without thinking about it?

That's what CheckTrap is for.

```python
import asyncio
from healthchecks_io import Client, AsyncClient, CheckCreate, CheckTrap

def run_my_thing_to_monitor():
    pass

async def main(check):
    client = AsyncClient(ping_key="ping_key")

    # works with async too, and the ping api and slugs
    async with CheckTrap(client, slug=check.slug) as ct:
        # when entering the context manager, sends a start ping to your check
        # Add custom logs to what gets sent to healthchecks. Reminder, only the first␣
↪10k bytes get saved
        ct.add_log("My custom log message")
        run_my_thing_to_monitor()

if __name__ == "__main__":
    client = Client(api_key="myapikey")

    # create a new check, or use an existing one already with just its uuid.
    check = await client.create_check(CreateCheck(name="New Check", tags="tag1 tag2")

    with CheckTrap(client, check.uuid):
        # when entering the context manager, sends a start ping to your check
        run_my_thing_to_monitor()

    asyncio.run(main())
```

8.2 Reference

8.2.1 py_healthchecks.io

Py Healthchecks.Io.

`class healthchecks_io.`**`AsyncClient`**(*api_key='', ping_key='', api_url='https://healthchecks.io/api/', ping_url='https://hc-ping.com/', api_version=1, client=None*)

A Healthchecks.io client implemented using httpx's Async methods.

Parameters

- **`api_key`** (*`str`*)
- **`ping_key`** (*`str`*)
- **`api_url`** (*`str`*)
- **`ping_url`** (*`str`*)
- **`api_version`** (*`int`*)
- **`client`** (*`AsyncClient | None`*)

`async` **`create_check`**(*new_check*)

Creates a new check and returns it.

With this API call, you can create both Simple and Cron checks: * To create a Simple check, specify the timeout parameter. * To create a Cron check, specify the schedule and tz parameters.

Parameters
`new_check` (`CheckCreate`) – New check you are wanting to create

Returns
check that was just created

Return type
Check

`async` **`delete_check`**(*check_id*)

Permanently deletes the check from the user's account.

check_id must be a uuid, not a unique id

Parameters
`check_id` (*`str`*) – check's uuid

Returns
the check just deleted

Return type
Check

Raises

- *`HCAPIAuthError`* – Raised when status_code == 401 or 403
- *`HCAPIError`* – Raised when status_code is 5xx
- *`CheckNotFoundError`* – Raised when status_code is 404
- *`HCAPIRateLimitError`* – Raised when status code is 429

async exit_code_ping(*exit_code*, *uuid=''*, *slug=''*, *data=''*)

Signals to Healthchecks.io that the job has failed.

Actively signaling a failure minimizes the delay from your monitored service failing to you receiving an alert.

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check's slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the "409 Conflict" HTTP status code and ignore the request.

Parameters

- **exit_code** (*int*) – Exit code to sent, int from 0 to 255
- **uuid** (*str*) – Check's UUID. Defaults to "".
- **slug** (*str*) – Check's Slug. Defaults to "".
- **data** (*str*) – Text data to append to this check. Defaults to "".

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns

success (true or false) and the response text

Return type

Tuple[bool, str]

async fail_ping(*uuid=''*, *slug=''*, *data=''*)

Signals to Healthchecks.io that the job has failed.

Actively signaling a failure minimizes the delay from your monitored service failing to you receiving an alert.

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check's slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the "409 Conflict" HTTP status code and ignore the request.

Parameters

- **uuid** (*str*) – Check's UUID. Defaults to "".
- **slug** (*str*) – Check's Slug. Defaults to "".
- **data** (*str*) – Text data to append to this check. Defaults to "".

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns
success (true or false) and the response text

Return type
Tuple[bool, str]

`async get_badges()`

Returns a dict of all tags in the project, with badge URLs for each tag.

Healthchecks.io provides badges in a few different formats: svg: returns the badge as a SVG document. json: returns a JSON document which you can use to generate a custom badge yourself. shields: returns JSON in a Shields.io compatible format. In addition, badges have 2-state and 3-state variations:

svg, json, shields: reports two states: "up" and "down". It considers any checks in the grace period as still "up". svg3, json3, shields3: reports three states: "up", "late", and "down".

The response includes a special * entry: this pseudo-tag reports the overal status of all checks in the project.

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *HCAPIRateLimitError* – Raised when status code is 429

Returns
Dictionary of all tags in the project with badges

Return type
Dict[str, *Badges*]

`async get_check(`*check_id*`)`

Get a single check by id.

check_id can either be a check uuid if using a read/write api key or a unique key if using a read only api key.

Parameters
check_id (*str*) – check's uuid or unique id

Returns
the check

Return type
Check

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403

- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404
- *HCAPIRateLimitError* – Raised when status code is 429

`async get_check_flips`(*check_id*, *seconds=None*, *start=None*, *end=None*)

Returns a list of "flips" this check has experienced.

A flip is a change of status (from "down" to "up," or from "up" to "down").

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404
- *BadAPIRequestError* – Raised when status_code is 400
- *HCAPIRateLimitError* – Raised when status code is 429

Parameters

- **check_id** (*str*) – check uuid
- **seconds** (*Optional[int], optional*) – Returns the flips from the last value seconds. Defaults to None.
- **start** (*Optional[int], optional*) – Returns flips that are newer than the specified UNIX timestamp. Defaults to None.
- **end** (*Optional[int], optional*) – Returns flips that are older than the specified UNIX timestamp. Defaults to None.

Returns

List of status flips for this check

Return type

List[*CheckStatuses*]

`async get_check_pings`(*check_id*)

Returns a list of pings this check has received.

This endpoint returns pings in reverse order (most recent first), and the total number of returned pings depends on the account's billing plan: 100 for free accounts, 1000 for paid accounts.

Parameters

check_id (*str*) – check's uuid

Returns

list of pings this check has received

Return type

List[*CheckPings*]

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404
- *HCAPIRateLimitError* – Raised when status code is 429

async **get_checks**(*tags=None*)

Get a list of checks from the healthchecks api.

Parameters
tags (*Optional[List[str]], optional*) – Filters the checks and returns only the checks that are tagged with the specified value. Defaults to None.

Raises

- *HCAPIAuthError* – When the API returns a 401, indicates an api key issue
- *HCAPIError* – When the API returns anything other than a 200 or 401
- *HCAPIRateLimitError* – Raised when status code is 429

Returns
[description]

Return type
List[*Check*]

async **get_integrations**()

Returns a list of integrations belonging to the project.

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *HCAPIRateLimitError* – Raised when status code is 429

Returns
List of integrations for the project

Return type
List[Optional[*Integration*]]

async **pause_check**(*check_id*)

Disables monitoring for a check without removing it.

The check goes into a "paused" state. You can resume monitoring of the check by pinging it.

check_id must be a uuid, not a unique id

Parameters
check_id (*str*) – check's uuid

Returns
the check just paused

Return type
Check

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404

async start_ping(*uuid=''*, *slug=''*, *data=''*)

Sends a “job has started!” message to Healthchecks.io.

Sending a “start” signal is optional, but it enables a few extra features: * Healthchecks.io will measure and display job execution times * Healthchecks.io will detect if the job runs longer than its configured grace time

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check’s slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the “409 Conflict” HTTP status code and ignore the request.

Parameters

- **uuid** (*str*) – Check’s UUID. Defaults to “”.
- **slug** (*str*) – Check’s Slug. Defaults to “”.
- **data** (*str*) – Text data to append to this check. Defaults to “”.

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has “not found” in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has “rate limited” in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns

success (true or false) and the response text

Return type

Tuple[bool, str]

async success_ping(*uuid=''*, *slug=''*, *data=''*)

Signals to Healthchecks.io that a job has completed successfully.

Can also be used to indicate a continuously running process is still running and healthy.

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check’s slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the “409 Conflict” HTTP status code and ignore the request.

Parameters

- **uuid** (*str*) – Check’s UUID. Defaults to “”.
- **slug** (*str*) – Check’s Slug. Defaults to “”.
- **data** (*str*) – Text data to append to this check. Defaults to “”.

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403

- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns
success (true or false) and the response text

Return type
Tuple[bool, str]

async **update_check**(*uuid*, *update_check*)

Updates an existing check.

If you omit any parameter in update_check, Healthchecks.io will leave its value unchanged.

Parameters

- **uuid** (*str*) – UUID for the check to update
- **update_check** (CheckCreate) – Check values you want to update

Returns
check that was just updated

Return type
Check

exception healthchecks_io.**BadAPIRequestError**

Thrown when an api request returns a 400.

class healthchecks_io.**Badges**(***, *svg*, *svg3*, *json_url*, *json3_url*, *shields*, *shields3*)

Object with the Badges urls.

Parameters

- **svg** (*str*)
- **svg3** (*str*)
- **json_url** (*str*)
- **json3_url** (*str*)
- **shields** (*str*)
- **shields3** (*str*)

classmethod **from_api_result**(*badges_dict*)

Converts a dictionary from the healthchecks api into a Badges object.

Parameters
badges_dict (*Dict[str, str]*)

Return type
Badges

`class healthchecks_io.Check`(**, unique_key=None, name, slug, tags=None, desc=None, grace, n_pings, status, last_ping=None, next_ping=None, manual_resume, methods=None, ping_url=None, update_url=None, pause_url=None, channels=None, timeout=None, uuid=None*)

Schema for a check object, either from a readonly api request or a rw api request.

Parameters

- **unique_key** (*str | None*)
- **name** (*str*)
- **slug** (*str*)
- **tags** (*str | None*)
- **desc** (*str | None*)
- **grace** (*int*)
- **n_pings** (*int*)
- **status** (*str*)
- **last_ping** (*datetime | None*)
- **next_ping** (*datetime | None*)
- **manual_resume** (*bool*)
- **methods** (*str | None*)
- **ping_url** (*str | None*)
- **update_url** (*str | None*)
- **pause_url** (*str | None*)
- **channels** (*str | None*)
- **timeout** (*int | None*)
- **uuid** (*str | None*)

`classmethod from_api_result`(*check_dict*)

Converts a dictionary from the healthchecks api into an Check object.

Parameters
check_dict (*Dict[str, Any]*)

Return type
Check

`classmethod validate_uuid`(*value, values*)

Tries to set the uuid from the ping_url.

Will return none if a read only token is used because it cannot retrieve the UUID of a check

Parameters

- **value** (*str | None*)
- **values** (*Dict[str, Any]*)

Return type
str | None

class healthchecks_io.CheckCreate(**, name='', tags='', desc='', timeout=86400, grace=3600, schedule=None, tz='UTC', manual_resume=False, methods='', channels=None, unique=[]*)

Pydantic object for creating a check.

Parameters

- **name** (*str | None*)
- **tags** (*str | None*)
- **desc** (*str | None*)
- **timeout** (*int | None*)
- **grace** (*int | None*)
- **schedule** (*str | None*)
- **tz** (*str | None*)
- **manual_resume** (*bool | None*)
- **methods** (*str | None*)
- **channels** (*str | None*)
- **unique** (*List[str | None] | None*)

classmethod validate_methods(*value*)

Validate that methods.

Parameters
value (*str*)

Return type
str

classmethod validate_schedule(*value*)

Validates that the schedule is a valid cron expression.

Parameters
value (*str*)

Return type
str

classmethod validate_tz(*value*)

Validates that the timezone is a valid timezone string.

Parameters
value (*str*)

Return type
str

classmethod validate_unique(*value*)

Validate unique list.

Parameters
value (*List[str | None]*)

Return type
List[str | None]

exception healthchecks_io.**CheckNotFoundError**

Thrown when getting a check returns a 404.

class healthchecks_io.**CheckPings**(*, *type*, *date*, *number_of_pings*, *scheme*, *remote_addr*, *method*, *user_agent*, *duration=None*)

A Pydantic schema for a check's Pings.

Parameters

- **type** (*str*)
- **date** (*datetime*)
- **number_of_pings** (*int*)
- **scheme** (*str*)
- **remote_addr** (*str*)
- **method** (*str*)
- **user_agent** (*str*)
- **duration** (*float | None*)

classmethod from_api_result(*ping_dict*)

Converts a dictionary from the healthchecks api into a CheckPings object.

Parameters
ping_dict (*Dict[str, str | int | datetime]*)

Return type
CheckPings

class healthchecks_io.**CheckStatuses**(*, *timestamp*, *up*)

A Pydantic schema for a check's Statuses.

Parameters

- **timestamp** (*datetime*)
- **up** (*int*)

class healthchecks_io.**CheckTrap**(*client*, *uuid=''*, *slug=''*, *suppress_exceptions=False*)

CheckTrap is a context manager to wrap around python code to communicate results to a Healthchecks check.

Parameters

- **client** (Client | AsyncClient)
- **uuid** (*str*)
- **slug** (*str*)
- **suppress_exceptions** (*bool*)

add_log(*line*)

Add a line to the context manager's log that is sent with the check.

Parameters
line (*str*) – String to add to the logs

Return type
None

class healthchecks_io.**CheckUpdate**(**, name=None, tags=None, desc='', timeout=None, grace=None, schedule=None, tz=None, manual_resume=None, methods=None, channels=None, unique=None*)

Pydantic object for updating a check.

Parameters

- **name** (*str | None*)
- **tags** (*str | None*)
- **desc** (*str | None*)
- **timeout** (*int | None*)
- **grace** (*int | None*)
- **schedule** (*str | None*)
- **tz** (*str | None*)
- **manual_resume** (*bool | None*)
- **methods** (*str | None*)
- **channels** (*str | None*)
- **unique** (*List[str | None] | None*)

class healthchecks_io.**Client**(*api_key='', ping_key='', api_url='https://healthchecks.io/api/', ping_url='https://hc-ping.com/', api_version=1, client=None*)

A Healthchecks.io client implemented using httpx's sync methods.

Parameters

- **api_key** (*str*)
- **ping_key** (*str*)
- **api_url** (*str*)
- **ping_url** (*str*)
- **api_version** (*int*)
- **client** (*Client | None*)

create_check(*new_check*)

Creates a new check and returns it.

With this API call, you can create both Simple and Cron checks: * To create a Simple check, specify the timeout parameter. * To create a Cron check, specify the schedule and tz parameters.

Parameters
new_check (CheckCreate) – New check you are wanting to create

Returns
check that was just created

Return type
Check

delete_check(*check_id*)

Permanently deletes the check from the user's account.

check_id must be a uuid, not a unique id

Parameters
`check_id` (*str*) – check's uuid

Returns
the check just deleted

Return type
checks.Check

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404

`exit_code_ping`(*exit_code*, *uuid=''*, *slug=''*, *data=''*)

Signals to Healthchecks.io that the job has failed.

Actively signaling a failure minimizes the delay from your monitored service failing to you receiving an alert.

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check's slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the "409 Conflict" HTTP status code and ignore the request.

Parameters

- **exit_code** (*int*) – Exit code to sent, int from 0 to 255
- **uuid** (*str*) – Check's UUID. Defaults to "".
- **slug** (*str*) – Check's Slug. Defaults to "".
- **data** (*str*) – Text data to append to this check. Defaults to ""

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns
success (true or false) and the response text

Return type
Tuple[bool, str]

fail_ping(*uuid=''*, *slug=''*, *data=''*)

Signals to Healthchecks.io that the job has failed.

Actively signaling a failure minimizes the delay from your monitored service failing to you receiving an alert.

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check's slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the "409 Conflict" HTTP status code and ignore the request.

Parameters

- **uuid** (*str*) – Check's UUID. Defaults to "".
- **slug** (*str*) – Check's Slug. Defaults to "".
- **data** (*str*) – Text data to append to this check. Defaults to ""

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns
success (true or false) and the response text

Return type
Tuple[bool, str]

get_badges()

Returns a dict of all tags in the project, with badge URLs for each tag.

Healthchecks.io provides badges in a few different formats: svg: returns the badge as a SVG document. json: returns a JSON document which you can use to generate a custom badge yourself. shields: returns JSON in a Shields.io compatible format. In addition, badges have 2-state and 3-state variations:

svg, json, shields: reports two states: "up" and "down". It considers any checks in the grace period
as still "up".

svg3, json3, shields3: reports three states: "up", "late", and "down".

The response includes a special * entry: this pseudo-tag reports the overal status of all checks in the project.

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx

Returns
Dictionary of all tags in the project with badges

Return type
Dict[str, *badges.Badges*]

`get_check`(*check_id*)

Get a single check by id.

check_id can either be a check uuid if using a read/write api key or a unique key if using a read only api key.

Parameters
`check_id` (*`str`*) – check's uuid or unique id

Returns
the check

Return type
checks.Check

Raises

- *`HCAPIAuthError`* – Raised when status_code == 401 or 403
- *`HCAPIError`* – Raised when status_code is 5xx
- *`CheckNotFoundError`* – Raised when status_code is 404

`get_check_flips`(*check_id*, *seconds=None*, *start=None*, *end=None*)

Returns a list of "flips" this check has experienced.

A flip is a change of status (from "down" to "up," or from "up" to "down").

Raises

- *`HCAPIAuthError`* – Raised when status_code == 401 or 403
- *`HCAPIError`* – Raised when status_code is 5xx
- *`CheckNotFoundError`* – Raised when status_code is 404
- *`BadAPIRequestError`* – Raised when status_code is 400

Parameters

- **`check_id`** (*`str`*) – check uuid
- **`seconds`** (*`Optional[int], optional`*) – Returns the flips from the last value seconds. Defaults to None.
- **`start`** (*`Optional[int], optional`*) – Returns flips that are newer than the specified UNIX timestamp. Defaults to None.
- **`end`** (*`Optional[int], optional`*) – Returns flips that are older than the specified UNIX timestamp. Defaults to None.

Returns
List of status flips for this check

Return type
List[*checks.CheckStatuses*]

`get_check_pings`(*check_id*)

Returns a list of pings this check has received.

This endpoint returns pings in reverse order (most recent first), and the total number of returned pings depends on the account's billing plan: 100 for free accounts, 1000 for paid accounts.

Parameters
`check_id` (*str*) – check's uuid

Returns
list of pings this check has received

Return type
List[*checks.CheckPings*]

Raises

- ***HCAPIAuthError*** – Raised when status_code == 401 or 403
- ***HCAPIError*** – Raised when status_code is 5xx
- ***CheckNotFoundError*** – Raised when status_code is 404

get_checks(*tags=None*)

Get a list of checks from the healthchecks api.

Parameters
`tags` (*Optional[List[str]], optional*) – Filters the checks and returns only the checks that are tagged with the specified value. Defaults to None.

Raises

- ***HCAPIAuthError*** – When the API returns a 401, indicates an api key issue
- ***HCAPIError*** – When the API returns anything other than a 200 or 401

Returns
[description]

Return type
List[*checks.Check*]

get_integrations()

Returns a list of integrations belonging to the project.

Raises

- ***HCAPIAuthError*** – Raised when status_code == 401 or 403
- ***HCAPIError*** – Raised when status_code is 5xx

Returns
List of integrations for the project

Return type
List[Optional[*integrations.Integration*]]

pause_check(*check_id*)

Disables monitoring for a check without removing it.

The check goes into a "paused" state. You can resume monitoring of the check by pinging it.

check_id must be a uuid, not a unique id

Parameters
`check_id` (*str*) – check's uuid

Returns
the check just paused

Return type
checks.Check

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404

start_ping(*uuid=''*, *slug=''*, *data=''*)

Sends a "job has started!" message to Healthchecks.io.

Sending a "start" signal is optional, but it enables a few extra features: * Healthchecks.io will measure and display job execution times * Healthchecks.io will detect if the job runs longer than its configured grace time

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check's slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the "409 Conflict" HTTP status code and ignore the request.

Parameters

- **uuid** (*str*) – Check's UUID. Defaults to "".
- **slug** (*str*) – Check's Slug. Defaults to "".
- **data** (*str*) – Text data to append to this check. Defaults to ""

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns

success (true or false) and the response text

Return type

Tuple[bool, str]

success_ping(*uuid=''*, *slug=''*, *data=''*)

Signals to Healthchecks.io that a job has completed successfully.

Can also be used to indicate a continuously running process is still running and healthy.

Can take a uuid or a slug. If you call with a slug, you much have a ping key set.

Check's slug is not guaranteed to be unique. If multiple checks in the project have the same name, they also have the same slug. If you make a Pinging API request using a non-unique slug, Healthchecks.io will return the "409 Conflict" HTTP status code and ignore the request.

Parameters

- **uuid** (*str*) – Check's UUID. Defaults to "".

- **slug** (*str*) – Check's Slug. Defaults to "".
- **data** (*str*) – Text data to append to this check. Defaults to ""

Raises

- *HCAPIAuthError* – Raised when status_code == 401 or 403
- *HCAPIError* – Raised when status_code is 5xx
- *CheckNotFoundError* – Raised when status_code is 404 or response text has "not found" in it
- *BadAPIRequestError* – Raised when status_code is 400, or if you pass a uuid and a slug, or if pinging by a slug and do not have a ping key set
- *HCAPIRateLimitError* – Raised when status code is 429 or response text has "rate limited" in it
- *NonUniqueSlugError* – Raused when status code is 409.

Returns
success (true or false) and the response text

Return type
Tuple[bool, str]

update_check(*uuid*, *update_check*)

Updates an existing check.

If you omit any parameter in update_check, Healthchecks.io will leave its value unchanged.

With this API call, you can create both Simple and Cron checks: * To create a Simple check, specify the timeout parameter. * To create a Cron check, specify the schedule and tz parameters.

Parameters

- **uuid** (*str*) – UUID for the check to update
- **update_check** (CheckCreate) – Check values you want to update

Returns
check that was just updated

Return type
Check

exception healthchecks_io.**HCAPIAuthError**

Thrown when we fail to auth to the Healthchecks api.

exception healthchecks_io.**HCAPIError**

API Exception for when we have an error with the healthchecks api.

exception healthchecks_io.**HCAPIRateLimitError**

Thrown when the api returns a rate limit response.

class healthchecks_io.**Integration**(*, *id*, *name*, *kind*)

Schema for an integration object.

Parameters

- **id** (*str*)
- **name** (*str*)
- **kind** (*str*)

classmethod from_api_result(*integration_dict*)

Converts a dictionary from the healthchecks api into an Integration object.

Parameters
integration_dict (*Dict[str, str]*)

Return type
Integration

exception healthchecks_io.**NonUniqueSlugError**

Thrown when the api returns a 409 when pinging.

exception healthchecks_io.**PingFailedError**

Thrown when a ping fails.

exception healthchecks_io.**WrongClientError**

Thrown when trying to use a CheckTrap with the wrong client type.

8.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

8.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.3.2 How to request a feature

Request features on the Issue Tracker.

8.3.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run py-healthchecks.io
```

8.3.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

8.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.4 Contributor Covenant Code of Conduct

8.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at andrew@.kz. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.5 MIT License

PYTHON MODULE INDEX

h

healthchecks_io, 20

INDEX

A

add_log() (*healthchecks_io.CheckTrap method*), 29
AsyncClient (*class in healthchecks_io*), 20

B

BadAPIRequestError, 26
Badges (*class in healthchecks_io*), 26

C

Check (*class in healthchecks_io*), 26
CheckCreate (*class in healthchecks_io*), 27
CheckNotFoundError, 28
CheckPings (*class in healthchecks_io*), 29
CheckStatuses (*class in healthchecks_io*), 29
CheckTrap (*class in healthchecks_io*), 29
CheckUpdate (*class in healthchecks_io*), 29
Client (*class in healthchecks_io*), 30
create_check() (*healthchecks_io.AsyncClient method*), 20
create_check() (*healthchecks_io.Client method*), 30

D

delete_check() (*healthchecks_io.AsyncClient method*), 20
delete_check() (*healthchecks_io.Client method*), 30

E

exit_code_ping() (*healthchecks_io.AsyncClient method*), 20
exit_code_ping() (*healthchecks_io.Client method*), 31

F

fail_ping() (*healthchecks_io.AsyncClient method*), 21
fail_ping() (*healthchecks_io.Client method*), 31
from_api_result() (*healthchecks_io.Badges class method*), 26
from_api_result() (*healthchecks_io.Check class method*), 27
from_api_result() (*healthchecks_io.CheckPings class method*), 29
from_api_result() (*healthchecks_io.Integration class method*), 36

G

get_badges() (*healthchecks_io.AsyncClient method*), 22
get_badges() (*healthchecks_io.Client method*), 32
get_check() (*healthchecks_io.AsyncClient method*), 22
get_check() (*healthchecks_io.Client method*), 33
get_check_flips() (*healthchecks_io.AsyncClient method*), 23
get_check_flips() (*healthchecks_io.Client method*), 33
get_check_pings() (*healthchecks_io.AsyncClient method*), 23
get_check_pings() (*healthchecks_io.Client method*), 33
get_checks() (*healthchecks_io.AsyncClient method*), 23
get_checks() (*healthchecks_io.Client method*), 34
get_integrations() (*healthchecks_io.AsyncClient method*), 24
get_integrations() (*healthchecks_io.Client method*), 34

H

HCAPIAuthError, 36
HCAPIError, 36
HCAPIRateLimitError, 36
healthchecks_io
    module, 20

I

Integration (*class in healthchecks_io*), 36

M

module
    healthchecks_io, 20

N

NonUniqueSlugError, 37

P

pause_check() (*healthchecks_io.AsyncClient method*), 24

pause_check() (*healthchecks_io.Client method*), 34
PingFailedError, 37

S

start_ping() (*healthchecks_io.AsyncClient method*), 24
start_ping() (*healthchecks_io.Client method*), 35
success_ping() (*healthchecks_io.AsyncClient method*), 25
success_ping() (*healthchecks_io.Client method*), 35

U

update_check() (*healthchecks_io.AsyncClient method*), 26
update_check() (*healthchecks_io.Client method*), 36

V

validate_methods() (*healthchecks_io.CheckCreate class method*), 28
validate_schedule() (*healthchecks_io.CheckCreate class method*), 28
validate_tz() (*healthchecks_io.CheckCreate class method*), 28
validate_unique() (*healthchecks_io.CheckCreate class method*), 28
validate_uuid() (*healthchecks_io.Check class method*), 27

W

WrongClientError, 37