NS1 Python SDK Documentation

Release 0.9.17

NSONE, Inc.

Jan 24, 2018

Contents

NS1.

CHAPTER 1

About

This package provides an SDK for accessing the NS1 DNS platform and includes both a simple NS1 REST API wrapper as well as a higher level interface for managing zones, records, data feeds, and more. It supports synchronous and asynchronous transports.

Both python 2.7 and 3.3 are supported.

Install with:

```
$ pip install ns1-python
```

CHAPTER 2

Quick Start

First, you'll need an API Key. To create one, login to the portal and click on the Account button in the top right. Select Settings & Users, then add a new API Key at the bottom.

Simple example:

```
from ns1 import NS1

api = NS1(apiKey='qACMD09OJXBxT7XOuRs8')
zone = api.createZone('example.com', nx_ttl=3600)
print(zone)
record = zone.add_A('honey', ['1.2.3.4', '5.6.7.8'])
print(record)
```

Note that all zone and record changes propagate in real time throughout the NS1 platform.

There are more examples in the examples directory.

CHAPTER 3

Contributions

We welcome contributions! Please fork on GitHub and submit a Pull Request.

CHAPTER 4

Contents

4.1 Features

- Extensive config system with support for multiple API keys
- High level interface to Zones and Records
- Low level REST wrapper for all other functionality
- Extendable transport system with synchronous and asynchronous transports

4.2 Configuration

Configuring the SDK can be done programmatically and/or via loading (and saving) simple JSON text configuration files. At a minimum, the NS1 API key to access the REST API must be specified.

4.2.1 Loading From a File

By default, configuration is loaded from the file `~/.nsone`; that is, a file called `.nsone` in the home directory of the user calling the script.

```
# to load an explicit configuration file:
api = NS1(configFile='/etc/ns1/api.json')
```

4.2.2 From an API Key

```
# to generate a configuration based on an api key
api = NS1(apiKey='qACMD09OJXBxT7XOuRs8')
```

4.2.3 JSON File Format

This example shows two different API keys. Which to use can be selected at runtime, see *ns1.config*

```
{
   "default_key": "account2",
   "verbosity": 5,
   "keys": {
        "account1": {
            "key": "qACMD09OJXBxT7XOuRs8",
            "desc": "account number 1",
            "writeLock": true
        },
        "account2": {
            "key": "qACMD09OJXBxT7XOwv9v",
            "desc": "account number 2",
            "writeLock": false
        },
   },
   "cli": {
       "output_format": "text"
   }
}
```

4.2.4 More

There are more examples in the config.py example. For the full Config object reference API, see *ns1.config*

4.3 Usage

There are many examples of usage in the examples directory.

CHAPTER 5

Reference

5.1 ns1 package

5.1.1 ns1.NS1

This top level object is used to initialize and coordinate access to the NS1 platform. With it, you create objects for accessing either the basic REST interface, or the high level objects such as Zone and Record.

class ns1.**NS1**(*apiKey=None*, *config=None*, *configFile=None*, *keyID=None*)
Create a new top level NS1 API object

Parameters

- **apiKey** (*str*) – if given, initialize config with this API key (obtainable via creation in NS1 portal)
- **config** (ns1.config.Config) – if given, uses a separately constructed and configured Config object
- **configFile** (*str*) – if given, load configuration from the given json configuration file
- **keyID** (*str*) – if given, use the specified key config in the multi-key configuration file

createZone(*zone*, *zoneFile=None*, *callback=None*, *errback=None*, ***kwargs*)
Create a new zone, and return an associated high level Zone object. Several optional keyword arguments are available to configure the SOA record.

If zoneFile is specified, upload the specific zone definition file to populate the zone with.

Parameters

- **zone** (*str*) – zone name, like 'example.com'
- **zoneFile** (*str*) – absolute path of a zone file
- **retry** (*int*) – retry time
- **refresh** (*int*) – refresh ttl

- **expiry** (*int*) – expiry ttl
- **nx_ttl** (*int*) – nxdomain TTL

Return type *ns1.zones.Zone*

datafeed()
Return a new raw REST interface to feed resources

Return type *ns1.rest.data.Feed*

datasource()
Return a new raw REST interface to datasource resources

Return type *ns1.rest.data.Source*

loadRecord(*domain*, *type*, *zone=None*, *callback=None*, *errback=None*, ***kwargs*)
Load an existing record into a high level Record object.

Parameters

- **domain** (*str*) – domain name of the record in the zone, for example 'myrecord'. You may leave off the zone, since it must be specified in the zone parameter
- **type** (*str*) – record type, such as 'A', 'MX', 'AAAA', etc.
- **zone** (*str*) – zone name, like 'example.com'

Return type *ns1.records*

loadZone(*zone*, *callback=None*, *errback=None*)
Load an existing zone into a high level Zone object.

Parameters **zone** (*str*) – zone name, like 'example.com'

Return type *ns1.zones.Zone*

monitors()
Return a new raw REST interface to monitors resources

Return type ns1.rest.monitoring.Monitors

notifylists()
Return a new raw REST interface to notify list resources

Return type ns1.rest.monitoring.NotifyLists

plan()
Return a new raw REST interface to account plan

Return type ns1.rest.account.Plan

records()
Return a new raw REST interface to record resources

Return type *ns1.rest.records.Records*

searchZone(*zone*, *q=None*, *has_geo=False*, *callback=None*, *errback=None*)
Search a zone for a given search query (e.g., for geological data, etc)

Parameters **zone** – NOT a string like loadZone - an already loaded ns1.zones.Zone, like one returned from loadZone

Returns

stats()
Return a new raw REST interface to stats resources

Return type *ns1.rest.stats.Stats*

zones()
Return a new raw REST interface to zone resources

Return type *ns1.rest.zones.Zones*

5.1.2 ns1.config

This object is used to configure the SDK and REST client. It handles multiple API keys via a simple selection mechanism (keyID).

Sample:

```
{
   "default_key": "account2",
   "verbosity": 5,
   "keys": {
        "account1": {
            "key": "qACMD09OJXBxT7XOuRs8",
            "desc": "account number 1",
            "writeLock": true
        },
        "account2": {
            "key": "qACMD09OJXBxT7XOwv9v",
            "desc": "account number 2",
            "writeLock": false
        }
   },
   "cli": {
       "output_format": "text"
   }
}
```

class ns1.config.**Config**(*path=None*)
A simple object for accessing and manipulating config files. These contains options and credentials for accessing the NS1 REST API. Config files are simple JSON text files. To set or retrieve vales, use the object like a dict.

Parameters **path** (*str*) – optional path. if given, try to load the given config file

API_VERSION = 'v1'

DEFAULT_CONFIG_FILE = '~/.nsone'

ENDPOINT = 'api.nsone.net'

PORT = 443

createFromAPIKey(*apikey*, *maybeWriteDefault=False*)
Create a basic config from a single API key

Parameters

- **apikey** (*str*) – NS1 API Key, as created in the NS1 portal
- **maybeWriteDefault** (*bool*) – If True and DEFAULT_CONFIG_FILE doesn't exist write out the resulting config there.

get(*item*, *default=None*)
Retrieve a value from the config object.

Parameters

- **item** (*str*) – Key to lookup
- **default** – Default value to return if the requested item doesn't exist

Returns Requested value, or *default* if it didn't exist

getAPIKey(*keyID=None*)
Retrieve the NS1 API Key for the given keyID

Parameters **keyID** (*str*) – optional keyID to retrieve, or current if not passed

Returns API Key for the given keyID

getCurrentKeyID()
Retrieve the current keyID in use.

Returns current keyID in use

getEndpoint()
Retrieve the NS1 API Endpoint URL that will be used for requests.

Returns URL of the NS1 API that will be used for requests

getKeyConfig(*keyID=None*)
Get key configuration specified by *keyID*, or current keyID.

Parameters **keyID** (*str*) – optional keyID to retrieve, or current if not passed

Returns a dict of the request (or current) key config

isKeyWriteLocked(*keyID=None*)
Determine if a key config is write locked.

Parameters **keyID** (*str*) – optional keyID to retrieve, or current if not passed

Returns True if the given (or current) keyID is writeLocked

loadFromDict(*d*)
Load config data from the given dictionary

Parameters **d** (*dict*) – Python dictionary containing configuration items

loadFromFile(*path*)
Load JSON config file from disk at the given path

Parameters **path** (*str*) – path to config file

loadFromString(*body*)
Load config data (i.e. JSON text) from the given string

Parameters **body** (*str*) – config data in JSON format

useKeyID(*keyID*)
Use the given API key config specified by *keyID* during subsequent API calls

Parameters **keyID** (*str*) – an index into the 'keys' maintained in this config

write(*path=None*)

Write config data to disk. If this config object already has a path, it will write to it. If it doesn't, one must be passed during this call.

Parameters **path** (*str*) – path to config file

exception ns1.config.**ConfigException**
Bases: exceptions.Exception

5.1.3 ns1.zones

Object representing a single DNS zone.

Note: Answers to a record (the *answers* kwarg) should be passed as one of the following four structures, depending on how advanced the configuration for the answer needs to be:

1. A single string that is coerced to a single answer with no other fields e.g. meta. For example: *"1.1.1.1"*
2. A list of single strings that is coerced to several answers with no other fields e.g. meta. For example: *["1.1.1.1", "2.2.2.2"]*
3. A list of lists. In this case there will be as many answers as are in the outer list, and the answers themselves are used verbatim from the inner list (e.g. may have MX style *[10, '1.1.1.1]*), but no other fields e.g. meta. You must use this form for MX records, and if there is only one answer it still must be wrapped in an outer list.
4. A list of dicts. In this case it expects the full rest model and passes it along unchanged. You must use this form for any advanced record config like meta data or data feeds.

```
# Example of an advanced answer configuration (list of dicts)
record = yield zone.add_A('record',
                          [{'answer': ['1.1.1.1'],
                            'meta': {
                                'up': False
                                }
                            },
                           {'answer': ['9.9.9.9'],
                            'meta': {
                                'up': True
                                }
                            }],
                          filters=[{'up': {}}])
```

class ns1.zones.**Zone**(*config*, *zone*)

Bases: object

High level object representing a Zone. In addition to the documented methods, there are magic methods allowing easy creation of records in this zone. Simply can 'add_TYPE' where TYPE is a valid DNS record type, such as add_A(). See examples for more information.

Create a new high level Zone object

Parameters

- **config** (ns1.config.Config) – config object
- **zone** (*str*) – zone name

cloneRecord(*existing_domain*, *new_domain*, *rtype*, *zone=None*, *callback=None*, *errback=None*)

Clone the given record to a new record such that their configs are identical.

Parameters

- **existing_domain** (*str*) – The existing record to clone
- **new_domain** (*str*) – The name of the new cloned record
- **rtype** (*str*) – DNS record type
- **zone** (*str*) – Optional zone name, if the new record should exist in a different zone than the original record.

Return type *ns1.records.Record*

Returns new Record

create(*zoneFile=None*, *callback=None*, *errback=None*, ***kwargs*)
Create a new zone. Pass a list of keywords and their values to configure. For the list of keywords available for zone configuration, see *ns1.rest.zones.Zones.INT_FIELDS* and *ns1.rest.zones.Zones.PASSTHRU_FIELDS* If zoneFile is passed, it should be a zone text file on the local disk that will be used to populate the created zone file.

createLinkToSelf(*new_zone*, *callback=None*, *errback=None*, ***kwargs*)
Create a new linked zone, linking to ourselves. All records in this zone will then be available as "linked records" in the new zone.

Parameters **new_zone** (*str*) – the new zone name to link to this one

Returns new Zone

delete(*callback=None*, *errback=None*)
Delete the zone and ALL records it contains.

linkRecord(*existing_domain*, *new_domain*, *rtype*, *callback=None*, *errback=None*, ***kwargs*)
Create a new linked record in this zone. These records use the configuration (answers, ttl, filters, etc) from an existing record in the NS1 platform.

Parameters

- **existing_domain** (*str*) – FQDN of the target record whose config should be used. Does not have to be in the same zone.
- **new_domain** (*str*) – Name of the new (linked) record. Zone name is appended automatically.
- **rtype** (*str*) – DNS record type, which must match the target record.

Return type *ns1.records.Record*

Returns new Record

load(*callback=None*, *errback=None*, *reload=False*)
Load zone data from the API.

loadRecord(*domain*, *rtype*, *callback=None*, *errback=None*)
Load a high level Record object from a domain within this Zone.

Parameters

- **domain** (*str*) – The name of the record to load
- **rtype** (*str*) – The DNS record type

Return type *ns1.records.Record*

Returns new Record

qps(*callback=None*, *errback=None*)
Return the current QPS for this zone

Return type dict

Returns QPS information

reload(*callback=None*, *errback=None*)
Reload zone data from the API.

search(*q=None*, *has_geo=False*, *callback=None*, *errback=None*)
Search within a zone for specific metadata. Zone must already be loaded.

update(*callback=None*, *errback=None*, ***kwargs*)
Update zone configuration. Pass a list of keywords and their values to update. For the list of keywords available for zone configuration, see *ns1.rest.zones.Zones.INT_FIELDS* and *ns1.rest.zones.Zones.PASSTHRU_FIELDS*

usage(*callback=None*, *errback=None*, ***kwargs*)
Return the current usage information for this zone

Return type dict

Returns usage information

exception ns1.zones.**ZoneException**
Bases: exceptions.Exception

5.1.4 ns1.records

Object representing a single DNS record in a zone of a specific type.

Note: Answers to a record (the *answers* kwarg) should be passed as one of the following four structures, depending on how advanced the configuration for the answer needs to be:

1. A single string that is coerced to a single answer with no other fields e.g. meta. For example: *"1.1.1.1"*
2. A list of single strings that is coerced to several answers with no other fields e.g. meta. For example: *["1.1.1.1", "2.2.2.2"]*
3. A list of lists. In this case there will be as many answers as are in the outer list, and the answers themselves are used verbatim from the inner list (e.g. may have MX style *[10, '1.1.1.1]*), but no other fields e.g. meta. You must use this form for MX records, and if there is only one answer it still must be wrapped in an outer list.
4. A list of dicts. In this case it expects the full rest model and passes it along unchanged. You must use this form for any advanced record config like meta data or data feeds.

```
# Example of an advanced answer configuration (list of dicts)
record = yield zone.add_A('record',
                          [{'answer': ['1.1.1.1'],
                            'meta': {
                                'up': False
                                }
                            },
                           {'answer': ['9.9.9.9'],
                            'meta': {
                                'up': True
                                }
                            }],
                          filters=[{'up': {}}])
```

class ns1.records.**Record**(*parentZone*, *domain*, *type*)
Bases: object

High level object representing a Record

Create a new high level Record

Parameters

- **parentZone** (ns1.zones.Zone) – the high level Zone parent object
- **domain** (*str*) – full domain name this record represents. if the domain does not end with the zone name, it is appended.
- **type** (*str*) – The DNS record type (A, MX, etc)

addAnswers(*answers*, *callback=None*, *errback=None*, ***kwargs*)
Add answers to the record.

Parameters **answers** – answers structure. See the class note on answer format.

create(*callback=None*, *errback=None*, ***kwargs*)
Create new record. Pass a list of keywords and their values to config. For the list of keywords available for zone configuration, see *ns1.rest.records.Records.INT_FIELDS*, *ns1.rest.records.Records.PASSTHRU_FIELDS*, *ns1.rest.records.Records.BOOL_FIELDS*

delete(*callback=None*, *errback=None*)
Delete the record from the zone, including all advanced configuration, meta data, etc.

load(*callback=None*, *errback=None*, *reload=False*)
Load record data from the API.

qps(*callback=None*, *errback=None*)
Return the current QPS for this record

Return type dict

Returns QPS information

reload(*callback=None*, *errback=None*)
Reload record data from the API.

update(*callback=None*, *errback=None*, ***kwargs*)
Update record configuration. Pass list of keywords and their values to update. For the list of keywords available for zone configuration, see *ns1.rest.records.Records.INT_FIELDS*, *ns1.rest.records.Records.PASSTHRU_FIELDS*, *ns1.rest.records.Records.BOOL_FIELDS*

usage(*callback=None*, *errback=None*, ***kwargs*)
Return the current usage information for this record

Return type dict

Returns usage information

exception ns1.records.**RecordException**
Bases: exceptions.Exception

5.1.5 ns1.rest

A thin layer over the NS1 REST API

exception ns1.rest.errors.**AuthException**(*message*, *response=None*, *body=None*)
Bases: *ns1.rest.errors.ResourceException*

exception ns1.rest.errors.**RateLimitException**(*message*, *response=None*, *body=None*, *by=None*, *limit=None*, *remaining=None*, *period=None*)
Bases: *ns1.rest.errors.ResourceException*

exception ns1.rest.errors.**ResourceException**(*message*, *response=None*, *body=None*)
Bases: exceptions.Exception

class ns1.rest.resource.**BaseResource**(*config*)

Parameters **config** (ns1.config.Config) – config object used to build requests

BOOL_FIELDS = []

DEFAULT_TRANSPORT = 'requests'

INT_FIELDS = []

PASSTHRU_FIELDS = []

class ns1.rest.data.**Feed**(*config*)

Bases: *ns1.rest.resource.BaseResource*

Parameters **config** (ns1.config.Config) – config object used to build requests

PASSTHRU_FIELDS = ['name', 'config']

ROOT = 'data/feeds'

create(*sourceid*, *name*, *config*, *callback=None*, *errback=None*, ***kwargs*)

delete(*sourceid*, *feedid*, *callback=None*, *errback=None*)

list(*sourceid*, *callback=None*, *errback=None*)

retrieve(*sourceid*, *feedid*, *callback=None*, *errback=None*)

update(*sourceid*, *feedid*, *callback=None*, *errback=None*, ***kwargs*)

class ns1.rest.data.**Source**(*config*)

Bases: *ns1.rest.resource.BaseResource*

Parameters **config** (ns1.config.Config) – config object used to build requests

PASSTHRU_FIELDS = ['name', 'config']

ROOT = 'data/sources'

create(*name*, *sourcetype*, *callback=None*, *errback=None*, ***kwargs*)

delete(*sourceid*, *callback=None*, *errback=None*)

list(*callback=None*, *errback=None*)

publish(*sourceid*, *data*, *callback=None*, *errback=None*)

retrieve(*sourceid*, *callback=None*, *errback=None*)

update(*sourceid*, *callback=None*, *errback=None*, ***kwargs*)

class ns1.rest.stats.**Stats**(*config*)

Bases: *ns1.rest.resource.BaseResource*

Parameters **config** (ns1.config.Config) – config object used to build requests

ROOT = 'stats'

qps(*zone=None*, *domain=None*, *type=None*, *callback=None*, *errback=None*)

usage(*zone=None*, *domain=None*, *type=None*, *callback=None*, *errback=None*, ***kwargs*)

class ns1.rest.records.**Records**(*config*)

Bases: *ns1.rest.resource.BaseResource*

Parameters **config** (ns1.config.Config) – config object used to build requests

BOOL_FIELDS = ['use_csubnet', 'override_ttl']

INT_FIELDS = ['ttl']

PASSTHRU_FIELDS = ['networks', 'meta', 'regions', 'link']

ROOT = 'zones'

create(*zone*, *domain*, *type*, *callback=None*, *errback=None*, ***kwargs*)

create_raw(*zone*, *domain*, *type*, *body*, *callback=None*, *errback=None*, ***kwargs*)

delete(*zone*, *domain*, *type*, *callback=None*, *errback=None*)

retrieve(*zone*, *domain*, *type*, *callback=None*, *errback=None*)

update(*zone*, *domain*, *type*, *callback=None*, *errback=None*, ***kwargs*)

class ns1.rest.zones.**Zones**(*config*)

Bases: *ns1.rest.resource.BaseResource*

Parameters **config** (ns1.config.Config) – config object used to build requests

BOOL_FIELDS = ['dnssec']

INT_FIELDS = ['ttl', 'retry', 'refresh', 'expiry', 'nx_ttl']

PASSTHRU_FIELDS = ['secondary', 'hostmaster', 'meta', 'networks', 'link']

ROOT = 'zones'

SEARCH_ROOT = 'search'

create(*zone*, *callback=None*, *errback=None*, ***kwargs*)

delete(*zone*, *callback=None*, *errback=None*)

import_file(*zone*, *zoneFile*, *callback=None*, *errback=None*, ***kwargs*)

list(*callback=None*, *errback=None*)

retrieve(*zone*, *callback=None*, *errback=None*)

search(*zone*, *q=None*, *has_geo=False*, *callback=None*, *errback=None*)

update(*zone*, *callback=None*, *errback=None*, ***kwargs*)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

N

P

Q

R

S

U

W

Z