Skytap REST API module Documentation

Release 1.2.0

Bill Wellington, Michael Knowles, Caleb Hawkins

January 10, 2017

Contents

Skytap is a set of modules that we use to manage aspects of our Skytap infrastructure.

CHAPTER 1

Usage

To use the module, you'll need to create at least two environment variables:

```
SKYTAP_USER=kermit.frog@sesamestreet.net
SKYTAP_TOKEN=79824879aeb2b34534e112d23a3c
```

Optionally, you can also add:

```
SKYTAP_LOG_LEVEL=20
```

This can be a number between 0-50 and corresponds to the logging module from Python:

- DEBUG: 0
- INFO: 10
- WARNING: 20
- ERROR: 30
- CRITICAL: 50

An easy way to set these variables is to create a .skytap file in your home directory (~/.skytap) with the variables in there:

```
export SKYTAP_USER=kermit.frog@sesamestreet.net
export SKYTAP_TOKEN=79824879aeb2b34534e112d23a3c
```

Then you can source the file:

```
source ~/.skytap
```

to load the variables, or add that same source command to your ~/.bash_profile or equivalent file to have it done automatically.

1.1 Via the command line

Most modules can be accessed directly from the command line to get simple information. This functionally returns the JSON from the Skytap API:

```
python -m skytap.Environments
python -m skytap.Users
```

You'll get back a JSON for the request, something like:

```
[
  {
    "id": "12345",
    "url": "https://cloud.skytap.com/users/12345",
    "login_name": "kermit.frog@sesamestreet.net",
    "first_name": "Kermit",
    "last_name": "The Frog",
    "title": "Master of Ceremonies",
    "email": "kermit.frog@sesamestreet.net",
    "created_at": "2012-01-02T12:43:05-08:00",
    "deleted": false
  }
]
```

If you only want a one item returned instead of the full list, you can get that from the command line as well:

```
python -m skytap.Environments 12345
python -m skytap.Quotas svm_hours
```

1.2 Via Python script

To use this, simply import it:

```
import skytap
```

Then you can access the resource groups of interest.

A simple example:

```
import skytap
users = skytap.Users()
for u in users:
    print(u.name + ' : ' + u.email)
```

This can also help automate running and suspending VMs:

```
import skytap
envs = skytap.Environments()

envs[123456].suspend()  # or .suspend(True) if you want the script to wait.
```

Doing this will, by default, add a note to the environment of it's action, so someone checking the environment can see why it's not running.

CHAPTER 2

Installation

Install this through pip:

```
pip install skytap
```

CHAPTER 3

Contributor list

- Bill Wellington github twitter blog
- Michael Knowles github twitter blog
- Caleb Hawkins github twitter blog

Contact us directly for questions.

CHAPTER 4

Objects

4.1 Skytap REST API wrapper

Skytap is a set of modules that we use to manage aspects of our Skytap infrastructure.

4.1.1 Usage

To use the module, you'll need to create at least two environment variables:

```
SKYTAP_USER=kermit.frog@sesamestreet.net
SKYTAP_TOKEN=79824879aeb2b34534e112d23a3c
```

Optionally, you can also add:

```
SKYTAP_LOG_LEVEL=20
```

This can be a number between 0-50 and corresponds to the logging module from Python:

- DEBUG: 0
- INFO: 10
- WARNING: 20
- ERROR: 30
- CRITICAL: 50

An easy way to set these variables is to create a .skytap file in your home directory (~/.skytap) with the variables in there:

```
export SKYTAP_USER=kermit.frog@sesamestreet.net
export SKYTAP_TOKEN=79824879aeb2b34534e112d23a3c
```

Then you can source the file:

```
source ~/.skytap
```

to load the variables, or add that same source command to your ~/.bash_profile or equivalent file to have it done automatically.

Via the command line

Most modules can be accessed directly from the command line to get simple information. This functionally returns the JSON from the Skytap API:

```
python -m skytap.Environments
python -m skytap.Users
```

You'll get back a JSON for the request, something like:

```
[
  {
    "id": "12345",
    "url": "https://cloud.skytap.com/users/12345",
    "login_name": "kermit.frog@sesamestreet.net",
    "first_name": "Kermit",
    "last_name": "The Frog",
    "title": "Master of Ceremonies",
    "email": "kermit.frog@sesamestreet.net",
    "created_at": "2012-01-02T12:43:05-08:00",
    "deleted": false
  }
]
```

If you only want a one item returned instead of the full list, you can get that from the command line as well:

```
python -m skytap.Environments 12345
python -m skytap.Quotas svm_hours
```

Via Python script

To use this, simply import it:

```
import skytap
```

Then you can access the resource groups of interest.

A simple example:

```
import skytap
users = skytap.Users()
for u in users:
    print(u.name + ' : ' + u.email)
```

This can also help automate running and suspending VMs:

```
import skytap
envs = skytap.Environments()

envs[123456].suspend()  # or .suspend(True) if you want the script to wait.
```

Doing this will, by default, add a note to the environment of it's action, so someone checking the environment can see why it's not running.

4.1.2 Installation

Install this through pip:

```
pip install skytap
```

4.1.3 Contributor list

- Bill Wellington github twitter blog
- Michael Knowles github twitter blog
- Caleb Hawkins github twitter blog

Contact us directly for questions.

4.2 Environments

Skytap API object wrapping Skytap Environments.

This roughly translates to the Skytap API call of /v2/configurations REST call, but gives us better access to the bits and pieces of the environments.

Accessing via command line

If accessed via the command line this will return the environments from Skytap in a JSON format:

```
python -m skytap.Environments
```

If you know the environment you want information on, you can also specify it directly. You can search by id or by a part of the environment name:

```
python -m skytap.Environments 12345
python -m skytap.Environments test
```

Additionally, you can search on some other crieria of a group to get a set you're looking for.

Runstate:

```
python -m skytap.Environments running  # or 'suspended',
    'stopped', or 'busy'
```

Region:

```
python -m skytap.Environments us-west
```

Accessing via Python

After you've installed `skytap` and added `import skytap` to your script, you can access the Skytap environments by the *`skytap.Environments`* object.

Example:

```
envs = skytap.Environments()
for e in envs:
    print (e.name)
```

Each environment has many things you can do with it - see the *`skytap.models.Environment`* object for actions you can take on an individual environment.

On the full list of environments, you can also get a vm count, svm count, get global storage, and delete environments. Each action is documented, below.

Environments can also perform any of the actions of other `SkytapGroup` objects. See the documentation on the *skytap.models.SkytapGroup* class for information there.

Note: Some pieces of a given environment, specifically *notes* and *user_data*, are only available via additional calls to the API. These fields will not exist when first creating the environments object, but any direct access to those fields will trigger the API call behind the scenes.

This is important if you're listing the entire contents (say, sending it to a JSON) - these fields won't be included if you haven't made that direct access.

This is by design to conserve API calls as most usage doesn't need or use those fields.

---

class skytap.Environments.**Environments**

Bases: *skytap.models.SkytapGroup.SkytapGroup*

Set of Skytap environments.

delete(*env*)

Delete a given environment.

> **Warning:** This is unrecoverable. Use with **extreme** caution.

Parameters **env** – The *Environment* to delete.

Returns True if the environment was deleted.

Return type bool

Raises `KeyError` – If `env` isn't in the Environments set.

Example:

```
envs = skytap.Environments()
target = envs[12345]
envs.delete(target)
```

storage()

Count the total amount of storage in use.

Returns Amount of storage used across all environments.

Return type int

Example:

```
envs = skytap.Environments()
print(envs.storage())
```

svms()

Count the total number of SVMs in use.

Returns Number of SVMs used across all environments.

Return type int

Example:

```
envs = skytap.Environments()
print(envs.svms())
```

vm_count()

Count the total number of VMs.

Returns Number of VMs used across all environments.

Return type int

Example:

```
envs = skytap.Environments()
print(envs.vm_count())
```

4.3 Groups

Skytap API object wrapping Skytap Groups.

This roughly translates to the Skytap API call of /v2/groups REST call, but gives us better access to the bits and pieces of the groups.

Accessing via command line

If accessed via the command line this will return the environments from Skytap in a JSON format:

```
python -m skytap.Groups
```

If you know the environment you want information on, you can also specify it directly. You can search by id or by a part of the environment name:

```
python -m skytap.Groups 12345
python -m skytap.Groups test
```

Accessing via Python

You can access the Skytap environments by the *skytap.Groups* object.

Example:

```
groups = skytap.Groups()
for g in groups:
    print(g.name)
```

Each group has many things you can do with it - see the *skytap.models.Group* object for actions you can take on an individual group.

On the full list of groups, you can also do a few other things:

- `add()`: add a new group.
- `delete()`: delete a group.

Environments can also perform any of the actions of other `SkytapGroup` objects. See the documentation on the *skytap.models.SkytapGroup* class for information there.

class `skytap.Groups.`**Groups**(*json_list=None*)

Bases: *skytap.models.SkytapGroup.SkytapGroup*

Set of Skytap groups.

Generally, access this through simply creating a *skytap.Groups* object.

Example:

```
groups = skytap.Groups()
for g in groups:
    print(g.name)
```

add(*group*, *description=''*)
Add one group.

Parameters

- **group** (*str*) – The group name to add.
- **description** (*str*) – The group description to add.

Returns The new group id from Skytap.

Return type int

Example:

```
groups = skytap.Groups()
new_group = groups.add('muppets', 'felt covered friends')
print(groups[new_group].name)
```

delete(*group*)
Delete a group.

Warning: This is unrecoverable. Use with caution.

Parameters **group** – The *Group* to delete.

Returns True if group deleted.

Return type bool

Raises TypeError – if group is not a *Group*

4.4 Projects

Support for Skytap API access to projects.

If accessed via the command line (`python -m skytap.Projects`) this will return the projects from Skytap in a JSON format.

class skytap.Projects.**Projects**
Bases: *skytap.models.SkytapGroup.SkytapGroup*

Set of Skytap projects.

Example:

4.5 Quotas

Support for Skytap API access to the company quotas.

If accessed via the command line (`python -m skytap.Quotas`) this will return the quotas from Skytap in a JSON format.

class skytap.Quotas.**Quotas**
Bases: *skytap.models.SkytapGroup.SkytapGroup*

Company/account quotas object.

Note: This code assumes that you have regional limits on your account. The return is different if you don't (see the /v2 API doc). We should get each piece of the return and sort it into type-and-region (whether you have

regional limits or not) and can then access things uniformly. Doing so will also require smartly accessing the API on demand more, since accounts with regional limits may require multiple calls to get the info desired.

4.6 Templates

Skytap API object wrapping Skytap templates.

This roughly translates to the Skytap API call of /v2/templates REST call, but gives us better access to the bits and pieces of the templates.

If accessed via the command line (`python -m skytap.Templates`) this will return the templates from Skytap in a JSON format.

class `skytap.Templates.`**`Templates`**
Bases: *skytap.models.SkytapGroup.SkytapGroup*

Set of Skytap templates.

Example

t = skytap.Templates() print len(t)

`storage`()
Count the total amount of storage in use.

`svms`()
Count the total number of SVMs in use.

`vm_count`()
Count the total number of VMs.

4.7 Users

Skytap API object wrapping Skytap users.

This roughly translates to the Skytap API call of /v2/users REST call, but gives us better access to the bits and pieces of the user.

If accessed via the command line (`python -m skytap.Users`) this will return the users from Skytap in a JSON format.

class `skytap.Users.`**`Users`**(*json_list=None*)
Bases: *skytap.models.SkytapGroup.SkytapGroup*

Set of Skytap users.

Example:

`add`(*login_name*, *email=None*)
Add one user.

Parameters

- **`login_name`** (*str*) – The login id of the account, usually an email.
- **`email`** (*str*) – The email of the account. If blank, will use login_name.

Returns The new user id from Skytap.

Return type int

Example:

```
users = skytap.Users()
new_user = users.add('kermit.frog@fulcrum.net')
print(users[new_user].login_name)
```

admins()
Count the numbers of admins.

delete(*user*, *transfer_user*)
Delete a user.

Warning: This is unrecoverable. Use with caution.

Parameters

- **user** – The user to delete (*User* or int).
- **transfer_user** – Transfer all assets to this user (*User* or int).

Returns True if user deleted.

Return type bool

Raises

- TypeError – if user or transfer_user is not a *User* or int.
- KeyError – If user or transfer_user isn't a user in the Users list.

4.8 VPNs

Skytap API object wrapping Skytap VPNs.

This roughly translates to the Skytap API call of /v2/vpns REST call, but gives us better access to the bits and pieces of the VPN.

If accessed via the command line (`python -m skytap.Vpns`) this will return the VPN information from Skytap in a JSON format.

class skytap.Vpns.**Vpns**
Bases: *skytap.models.SkytapGroup.SkytapGroup*

Set of Skytap VPNs.

Example

v = skytap.Vpns() print len(v)

4.9 Object Models

4.9.1 Environment module

Support for an Environment resource in Skytap.

In nearly every case, you'll access an Environment via the *Environments* object:

```
envs = skytap.Environments()
for environment in envs:
    print (environment.name)
```

You can access anything from an environment that Skytap includes in their API. Most of these can be access directly as attributes of the given Environment object:

```
environment = skytap.Environments()[12345]
print(environment.name)
print(environment.json)
```

Some data conversions are handled for you. Specifically:

- Dates are converted into `datetime` objects, like `created_at`.
- The `vms` list is loaded into a *skytap.models.Vms* class.
- The `notes` are put into a *skytap.models.Notes* class.
- The `user_data` is put into a *skytap.models.UserData* class.

There's also the ability to change the runstate of the environment through the function `change_state()`:

```
environment = skytap.Environments()[12345]
environment.change_state('suspended')

# Passing `True` will wait for the suspend to complete
# before returning to the script:
environment.change_state('suspended', True)
```

The various state change options also have easy aliases available to them:

- `run()`
- `halt()`
- `suspend()`
- `reset()`
- `stop()`

Passing *True* to any of these will also cause the script to wait until the action is completed by Skytap.

Note: Some pieces of a given environment, specifically *notes* and *user_data*, are only available via additional calls to the API. These fields will not exist when first creating the environments object, but any direct access to those fields will trigger the API call behind the scenes.

This is important if you're listing the entire contents (say, sending it to a JSON) - these fields won't be included if you haven't made that direct access.

This is by design to conserve API calls as most usage doesn't need or use those fields.

---

class `skytap.models.Environment.`**`Environment`**(*env_json*)

Bases: *skytap.models.SkytapResource.SkytapResource*, `skytap.framework.Suspendable.Suspendab`

One Skytap environment.

`__getattr__`(*key*)

Load values for anything that doesn't get loaded by default.

For user_data and notes, a secondary API call is needed. Only make that call when the info is requested.

delete()
Delete the environment.

In general, it'd seem wise not to do this very often.

4.9.2 Group module

Support for Skytap groups.

class skytap.models.Group.**Group**(*initial_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One Skytap Group.

__getattr__(*key*)
Load values for anything that doesn't get loaded by default.

For user_data and notes, a secondary API call is needed. Only make that call when the info is requested.

add_user(*user*)
Add a `User` to the group.

Parameters **user** (*int*) – id of the user to add.

Raises

- `TypeError` – If user is not an `int`.
- `KeyError` – If user is not in `Users` list.

Returns True if the user was added.

Return type bool

Example

```
>>> groups = skytap.Groups()
>>> users = skytap.Users()
>>> for u in users:
...     groups[12345].add(u.id)
```

delete()
Delete the group.

remove_user(*user*)
Remove a `user` from the group.

Parameters **user** (*int*) – id of the user to remove.

Raises

- `TypeError` – If user is not an `int`.
- `KeyError` – If user is not in `Users` list.

Returns True if the user was removed.

Return type bool

Example

```
>>> groups = skytap.Groups()
>>> groups[1234].remove_user(12345)
```

4.9.3 Note module

Support for a single note in a Skytap environment or vm.

class skytap.models.Note.**Note**(*note_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One note.

__str__()
Represent the Note as a string.

4.9.4 Notes module

Support for notes that are attached to VMs and environments.

class skytap.models.Notes.**Notes**(*note_json*, *env_url*)
Bases: *skytap.models.SkytapGroup.SkytapGroup*

A collection of notes.

add(*note*)
Add one note.

Parameters **note** (*str*) – The note text to add.

Returns The response from Skytap, typically the new note.

Return type str

delete(*note*)
Delete one note.

Parameters **note** – The *Note* to delete.

Returns The response from Skytap.

Return type str

Raises TypeError – If note is not a Note object.

delete_all()
Delete all notes.

Returns count of deleted notes.

Return type int

Use with care!

newest()
Return the newest note.

Returns The newest note.

Return type *Note*

oldest()
Return the oldest note.

Returns The oldest note.

Return type *Note*

Used most often to delete the oldest note.

Example:

```
notes = skytap.Environtments().first.notes
print(notes.oldest().text)
# notes.delete(notes.oldest())  # most common use case.
```

refresh()
Refresh the notes.

Raises KeyError – if the Notes object doesn't have a url attribute for some reason.

Go back to Skytap and get the notes again. Useful when you've changed the notes and to make sure you're current.

4.9.5 Project module

Support for Projects.

class skytap.models.Project.**Project**(*project_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One Skytap project.

4.9.6 Quota module

Support for Skytap quotas.

class skytap.models.Quota.**Quota**(*quota_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One piece of quota information.

__str__()
Represent object as a string.

4.9.7 SkytapGroup module

Base object to handle groups of Skytap objects.

class skytap.models.SkytapGroup.**SkytapGroup**
Bases: skytap.framework.ApiClient.ApiClient, six.Iterator

Base object for use with Skytap resource groups.

A SkytapGroup is essentially a set of SkytapResource objects. This allows us to more easily interact with 'VMs' as collections of VM objects. A list or dictionary could do much of this, but this allows us to also batch API calls intelligently (since many calls return a 'group' of data) and run other operations on multiple objects, when appropriate, like doing something across every VM inside an environment.

__contains__(*key*)
Check if the object contains an element.

__getitem__(*key*)
Return a data element from self.data.

__iter__()
Allow this object to be iterated over.

__len__()
Get length of the object.

__next__()
Get the next item in iteration.

__str__()
Represent the group as a string.

It'd be good to consider something more clever here, but returning the object back as a JSON also doesn't seem unreasonable as a way to make this data accessible to other processes.

find(*search*)
Return a list of objects, based on the search criteria.

This looks for matching ids if the search is a number, or searches the name if search is a string.

Parameters **search** (*int or str*) – What to search for.

Returns Any environments matching the search criteria.

Return type List

Example

```
>>> envs = skytap.Environments().search('testing')
```

first()
Return the first record in the list.

Mainly used to get a single arbitrary object for testing.

Returns An object from the list.

Return type *SkytapResource*

json()
Convert our list into a json.

keys()
Return the keys from the group list.

load_list_from_api(*url*, *target*, *params=None*)
Load something from the Skytap API and fill this object.

Parameters

- **url** (*str*) – The Skytap URL to load ('/v2/users').
- **target** – The *SkytapResource* type to load (For example: 'User')
- **params** (*dict*) – Any URL parameters to add to URL.

This should look like, in the child object:

```
self.load_list_from_api('/v2/projects', Project)
```

load_list_from_json(*json_list*, *target*, *url=None*, *params=None*)
Load items from a json list and fill this object.

Parameters

- **json_list** (*list*) – The list to load the items from.
- **target** – The *SkytapResource* type to load (for example: 'User')

This should look like, in the child object:

main(*argv*)
What to do when called from the command line.

This function is usually accessed via the command line:

```
python -m skytap.Environments
```

but can be used to return quick sets of formatted JSON:

```
>>> print(skytap.Environments().main())
```

Anything passed to the function will be searched for:

```
python -m skytap.Users fozzy
```

and:

```
>>> print(skytap.Users().main('scooter'))
```

Parameters **argv** (*list*) – Command line arguments

Returns Formatted JSON of the request.

Return type str

refresh()
Reload our data.

4.9.8 SkytapResource module

Base class for all Skytap Resources.

class skytap.models.SkytapResource.**SkytapResource**(*initial_json*)
Bases: object

Represents one Skytap Resource - a VM, Environment, User, whatever.

__contains__(*key*)
Check if this resource has a particular key in its data.

__eq__(*other*)
Compare the resource to another one, helpful for sorting.

__getattr__(*key*)
Access custom attributes.

This allows us to access members of self.data as if they're attributes of the resource. This transparently extends the object with all of the keys returned from the Skytap API without each resource having to do anything special except for exception cases.

__gt__(*other*)
Compare the resource to another one, helpful for sorting.

__hash__()
Represent the resource as a hash.

__int__()
Represent the resource as an int.

__lt__(*other*)
Compare the resource to another one, helpful for sorting.

__str__()
Represent the resource as a string.

json()
Convert the object to JSON.

refresh()
Refresh the data in our object, if we have a URL to pull from.

4.9.9 Template module

Support for an Template resource in Skytap.

class skytap.models.Template.**Template**(*tmp_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One Skytap template.

__getattr__(*key*)
Load values for anything that doesn't get loaded by default.

For user_data, a secondary API call is needed. Only make that call when the info is requested.

4.9.10 User module

Support for a User resource in Skytap.

class skytap.models.User.**User**(*user_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One Skytap User resource.

delete(*transfer_user*)
Delete the user.

4.9.11 UserData module

Support for the UserData resource in Skytap.

Specifically, this is for custom ('user data') that's applied to an environment or VM. This data can be text or, in the context of using it with this Skytap script, it can also be JSON or YAML and will then be re-parsed.

This allows users to put data into a VM user data block and it'll filter down and be accessible to this script. We use this to expose variables to the user like shutdown time and other automation pieces.

class skytap.models.UserData.**UserData**(*contents*, *env_url*)
Bases: *skytap.models.SkytapResource.SkytapResource*

UserData object to handle custom user data for a Skytap object.

This typically would be for a VM or Environment.

__str__()
Express the userdata as a string.

add(*key*, *value*)
Add value to environment's userdata.

Parameters

- **key** (*str*) – The name of the value's key.
- **value** (*str*) – The value to add.

Returns The response from Skytap, or "{}".

Return type str

add_line(*text*, *line=-1*)
Add line to environment's userdata.

Parameters

- **text** (*str*) – line of text to be added. (Required)
- **line** (*int*) – line number to add to. If too large, default to last.

Returns The response from Skytap.

Return type str

delete(*key*)
Delete key/value from environment's userdata.

Parameters **key** (*str*) – The name of key to delete, along with value

Returns The response from Skytap, or "{}".

Return type str

delete_line(*line*)
Delete line from environment's userdata.

Parameters **line** (*int*) – line number to delete.

Returns The response from Skytap.

Return type str

get_line(*line*)
Return content of line from environment's userdata.

Parameters **line** (*int*) – line number to get.

Returns The content of the line, or "".

Return type str

4.9.12 Vm module

Support for a VM resource in Skytap.

class skytap.models.Vm.**Vm**(*vm_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*, skytap.framework.Suspendable.Suspendab

One Skytap VM.

__getattr__(*key*)
Load values for anything that doesn't get loaded by default.

For user_data, notes, and interfaces, a secondary API call is needed. Only make that call when the info is requested.

delete()
Delete a VM.

In general, it'd seem wise not to do this very often.

4.9.13 Vms module

Support for Skytap VMs.

class skytap.models.Vms.**Vms**(*vms_json*, *env_url*)
Bases: *skytap.models.SkytapGroup.SkytapGroup*

A list of VMs.

4.9.14 Vpn module

Support for Skytap VPNs.

class skytap.models.Vpn.**Vpn**(*vpn_json*)
Bases: *skytap.models.SkytapResource.SkytapResource*

One Skytap VPN object.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

S

Symbols

A

D

E

F

G

J

K

L

M

N

O

P

Q

R

S

T

U

V