PyChef Documentation

Release 0.3.0

Noah Kantrowitz

Mar 24, 2017

Contents

CHAPTER 1

Getting Started

The first thing you have to do is load your Chef server credentials in to a *ChefAPI* object. The easiest way to do this is with *autoconfigure()*:

```
import chef
api = chef.autoconfigure()
```

Then we can load an object from the Chef server:

```
node = chef.Node('node_1')
```

And update it:

```
node.run_list.append('role[app]')
node.save()
```

API Reference

Chef API Interface

class chef.**ChefAPI**(*url*, *key*, *client*, *version='0.10.8'*, *headers={}*, *ssl_verify=True*)

The ChefAPI object is a wrapper for a single Chef server.

The API stack

PyChef maintains a stack of *ChefAPI* objects to be use with other methods if an API object isn't given explicitly. The first ChefAPI created will become the default, though you can set a specific default using *ChefAPI.set_default()*. You can also use a ChefAPI as a context manager to create a scoped default:

```
with ChefAPI('http://localhost:4000', 'client.pem', 'admin'):
    n = Node('web1')
```

---

classmethod **from_config_file**(*path*)
Load Chef API paraters from a config file. Returns None if the config can't be used.

static **get_global**()
Return the API on the top of the stack.

set_default()
Make this the default API in the stack. Returns the old default if any.

chef.**autoconfigure**(*base_path=None*)
Try to find a knife or chef-client config file to load parameters from, starting from either the given base path or the current working directory.

The lookup order mirrors the one from Chef, first all folders from the base path are walked back looking for .chef/knife.rb, then ~/.chef/knife.rb, and finally /etc/chef/client.rb.

The first file that is found and can be loaded successfully will be loaded into a *ChefAPI* object.

Nodes

class chef.**Node**(*name*, *api=None*, *skip_load=False*)
A Chef node object.

The Node object can be used as a dict-like object directly, as an alias for the *attributes* data:

```
>>> node = Node('name')
>>> node['apache']['log_dir']
'/var/log/apache2'
```

New in version 0.1.

attributes
NodeAttributes corresponding to the composite of all precedence levels. This only uses the stored data on the Chef server, it does not merge in attributes from roles or environments on its own.

```
>>> node.attributes['apache']['log_dir']
'/var/log/apache2'
```

run_list
The run list of the node. This is the unexpanded list in `type[name]` format.

```
>>> node.run_list
['role[base]', 'role[app]', 'recipe[web]']
```

chef_environment
The name of the Chef *Environment* this node is a member of. This value will still be present, even if communicating with a Chef 0.9 server, but will be ignored.

New in version 0.2.

default
NodeAttributes corresponding to the `default` precedence level.

normal
NodeAttributes corresponding to the `normal` precedence level.

override
NodeAttributes corresponding to the `override` precedence level.

automatic
NodeAttributes corresponding to the `automatic` precedence level.

create(*name*, *api=None*, ***kwargs*)
Create a new object of this type. Pass the initial value for any attributes as keyword arguments.

delete(*api=None*)
Delete this object from the server.

list(*api=None*)
Return a `ChefQuery` with the available objects of this type.

save(*api=None*)
Save this object to the server. If the object does not exist it will be created.

class `chef.node.`**NodeAttributes**(*search_path=[]*, *path=None*, *write=None*)
A collection of Chef *Node* attributes.

Attributes can be accessed like a normal python `dict`:

```
print node['fqdn']
node['apache']['log_dir'] = '/srv/log'
```

When writing to new attributes, any dicts required in the hierarchy are created automatically.

New in version 0.1.

get_dotted(*key*)
Retrieve an attribute using a dotted key path. A dotted path is a string of the form *'foo.bar.baz'*, with each `.` separating hierarcy levels.

Example:

```
node.attributes['apache']['log_dir'] = '/srv/log'
print node.attributes.get_dotted('apache.log_dir')
```

has_dotted(*key*)
Check if a given dotted key path is present. See *get_dotted()* for more information on dotted paths.

New in version 0.2.

set_dotted(*key*, *value*)
Set an attribute using a dotted key path. See *get_dotted()* for more information on dotted paths.

Example:

```
node.attributes.set_dotted('apache.log_dir', '/srv/log')
```

Roles

class `chef.`**Role**(*name*, *api=None*, *skip_load=False*)
A Chef role object.

create(*name*, *api=None*, ***kwargs*)
Create a new object of this type. Pass the initial value for any attributes as keyword arguments.

delete(*api=None*)
Delete this object from the server.

list(*api=None*)
Return a `ChefQuery` with the available objects of this type.

save(*api=None*)
Save this object to the server. If the object does not exist it will be created.

Data Bags

class chef.**DataBag**(*name*, *api=None*, *skip_load=False*)
A Chef data bag object.

Data bag items are available via the mapping API. Evaluation works in the same way as `ChefQuery`, so requesting only the names will not cause the items to be loaded:

```
bag = DataBag('versions')
item = bag['web']
for name, item in six.iteritems(bag):
    print item['qa_version']
```

create(*name*, *api=None*, ***kwargs*)
Create a new object of this type. Pass the initial value for any attributes as keyword arguments.

delete(*api=None*)
Delete this object from the server.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iterkeys() → an iterator over the keys of D

itervalues() → an iterator over the values of D

keys() → list of D's keys

list(*api=None*)
Return a `ChefQuery` with the available objects of this type.

save(*api=None*)
Save this object to the server. If the object does not exist it will be created.

values() → list of D's values

class chef.**DataBagItem**(*bag*, *name*, *api=None*, *skip_load=False*)
A Chef data bag item object.

Data bag items act as normal dicts and can contain arbitrary data.

bag
The *DataBag* this item is a member of.

clear() → None. Remove all items from D.

classmethod create(*bag*, *name*, *api=None*, ***kwargs*)
Create a new data bag item. Pass the initial value for any keys as keyword arguments.

delete(*api=None*)
Delete this object from the server.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iterkeys() → an iterator over the keys of D

itervalues() → an iterator over the values of D

keys() → list of D's keys

list(*api=None*)
Return a `ChefQuery` with the available objects of this type.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

save(*api=None*)
Save this object to the server. If the object does not exist it will be created.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → list of D's values

Environments

class `chef.`**Environment**(*name*, *api=None*, *skip_load=False*)
A Chef environment object.

New in version 0.2.

create(*name*, *api=None*, ***kwargs*)
Create a new object of this type. Pass the initial value for any attributes as keyword arguments.

delete(*api=None*)
Delete this object from the server.

list(*api=None*)
Return a `ChefQuery` with the available objects of this type.

save(*api=None*)
Save this object to the server. If the object does not exist it will be created.

Search

class `chef.`**Search**(*index*, *q='*:*'*, *rows=1000*, *start=0*, *api=None*)
A search of the Chef index.

The only required argument is the index name to search (eg. node, role, etc). The second, optional argument can be any Solr search query, with the same semantics as Chef.

Example:

```
for row in Search('node', 'roles:app'):
    print row['roles']
    print row.object.name
```

New in version 0.1.

count(*value*) → integer – return number of occurrences of value

Fabric Integration

chef.fabric.**chef_roledefs**(*api=None, hostname_attr=['cloud.public_hostname', 'fqdn'], environment=<object object>*)

Build a Fabric roledef dictionary from a Chef server.

Example:

```
from fabric.api import env, run, roles
from chef.fabric import chef_roledefs

env.roledefs = chef_roledefs()

@roles('web_app')
def mytask():
    run('uptime')
```

`hostname_attr` can either be a string that is the attribute in the chef node that holds the hostname or IP to connect to, an array of such keys to check in order (the first which exists will be used), or a callable which takes a *Node* and returns the hostname or IP to connect to.

To refer to a nested attribute, separate the levels with `'.'` e.g. `'ec2.public_hostname'`

`environment` is the Chef *Environment* name in which to search for nodes. If set to `None`, no environment filter is added. If set to a string, it is used verbatim as a filter string. If not passed as an argument at all, the value in the Fabric environment dict is used, defaulting to `'_default'`.

Note: `environment` must be set to `None` if you are emulating Chef API version 0.9 or lower.

New in version 0.1.

New in version 0.2: Support for iterable and callable values for the‘‘hostname_attr‘‘ argument, and the `environment` argument.

chef.fabric.**chef_environment**(*name*, *api=None*)

A Fabric task to set the current Chef environment context.

This task works alongside *chef_roledefs()* to set the Chef environment to be used in future role queries.

Example:

```
from chef.fabric import chef_environment, chef_roledefs
env.roledefs = chef_roledefs()
```

```
$ fab env:production deploy
```

The task can be configured slightly via Fabric `env` values.

`env.chef_environment_task_alias` sets the task alias, defaulting to “env”. This value must be set **before** *chef.fabric* is imported.

`env.chef_environment_validate` sets if *Environment* names should be validated before use. Defaults to True.

New in version 0.2.

chef.fabric.**chef_query**(*query, api=None, hostname_attr=['cloud.public_hostname', 'fqdn'], environment=<object object>*)

A decorator to use an arbitrary Chef search query to find nodes to execute on.

This is used like Fabric's `roles()` decorator, but accepts a Chef search query.

Example:

```
from chef.fabric import chef_query

@chef_query('roles:web AND tags:active')
@task
def deploy():
    pass
```

New in version 0.2.1.

chef.fabric.**chef_tags**(**tags, **kwargs*)

A decorator to use Chef node tags to find nodes to execute on.

This is used like Fabric's `roles()` decorator, but accepts a list of tags.

Example:

```
from chef.fabric import chef_tags

@chef_tags('active', 'migrator')
@task
def migrate():
    pass
```

New in version 0.2.1.

Opscode Authentication protocol

The Opscode authentication protocol is a specification for an HTTP authentication method using RSA signatures. It is used with chef-server-api as well as the Opscode Platform service.

Keys

Every client to a Chef server requires an RSA private key. These are generated by the server (or Platform) and should be stored securely. Keys must be in PEM format as defined by OpenSSL.

Headers

Each request must include 5 headers:

X-Ops-Sign Must be `version=1.0`.

X-Ops-Userid The name of the API client.

X-Ops-Timestamp The current time. See *Timestamp*.

X-Ops-Content-Hash The hash of the content of the request. See *Hashing*.

X-Ops-Authorization-$N The lines of the RSA signature. See *Signature*.

Canonicalization

Rules for canonicalizing request data.

HTTP Method

HTTP methods must be capitalized. Examples:

```
GET
POST
```

Timestamp

All timestamps must in ISO 8601 format using `T` as the separator. The timezone must be UTC, using `Z` as the indicator. Examples:

```
2010-12-04T15:47:49Z
```

Path

The path component of the URL must not have consecutive `/` characters. If it is not the root path (`^/$`), it must not end with a `/` character. Example:

```
/
/nodes
/nodes/example.com
```

Hashing

All hashes are Base64-encoded SHA1. The Base64 text must have line-breaks every 60 characters. The Base64 alphabet must be the standard alphabet defined in RFC 3548 (`+/=`).

Signature

The `X-Ops-Authorization-$N` headers must be a Base64 hash of the output of `RSA_private_encrypt`. Each line of the Base64 output is a new header, with the numbering starting at 1.

Base String

The signature base string is defined as:

```
Method:<HTTP method>\n
Hashed Path:<hashed path>\n
X-Ops-Content-Hash:<hashed_body>\n
X-Ops-Timestamp:<timestamp>\n
X-Ops-UserId:<client name>
```

All values must be canonicalized using the above rules.

Python Module Index

C

Index

P

R

S

U

V