NetScaler API Documentation

Release 0.2.3

Jathan McCollum

September 14, 2013

CONTENTS

CHAPTER

ONE

API DOCUMENTATION

CHAPTER

TWO

CHANGELOG

2.1 0.2.3

- #2: Fixed a typo in the call to `InteractionError`.
- Documentation improvments

2.2 0.2.2

- #1: Fixed a bug where setup.py was crashing with an `ImportError` when importing the version string from `netscaler.py` in the case where suds was not installed.

2.3 0.2.1

- Replaced all usage of dictionaries passed by reference in examples with keyword args.

2.4 0.2

- Added setup.py
- Added examples
- Added is_readonly() method to validate commands.
- Added save() shortcut method to self.client.service.savensconfig()
- Implemented autosave whenever a command executed ia run() is not read-only (Autosave can be disabled by passing autosave=False)

2.5 0.1

- Initial release

CHAPTER

THREE

LICENSE

CHAPTER

FOUR

SUMMARY

NetScaler API is a Python interface for interacting with Citrix NetScaler application delivery controllers, utilizing the SOAP API to execute commands.

CHAPTER

FIVE

DEPENDENCIES

:python-suds: Lightweight SOAP client

CHAPTER

SIX

EXAMPLE

Pass any kwargs to init that you would to the `suds.client.Client` constructor. A little bit of magic is performed with the `ImportDoctor` to cover missing types used in the WSDL.

- If you `specify wsdl`, this file will be pulled from the default http URL
- If you `specify wsdl_url`, it will override the wsdl file. Local
- `file://` URLs work just fine.

To save time for re-usable code, it is a good idea subclassing this to create methods for commonly used commands in your application:

```
class MyAPI(API):
    def change_password(self, username, newpass):
        return self.run("setsystemuser_password", username=username,
                        password=newpass)
```

In a script:

```
import netscaler

if __name__ == '__main__':
    netscaler.DEBUG = True
    wsdl_url = 'file:///home/j/jathan/sandbox/NSUserAdmin.wsdl'
    client = netscaler.API('nos', username='nsroot', password='nsroot', wsdl_url=wsdl_url)
    print client.logged_in
```

CHAPTER

SEVEN

DOCUMENTATION

7.1 API Documentation

Please review the *API Documentation*.

7.2 Changelog

Please review the *Changelog*.

7.3 License

Please review the *License*.

CHAPTER

EIGHT

PERFORMANCE

The default NetScaler WSDL is massive and is undoubtedly the most comprehensive SOAP API I have ever worked with. It is 2.5M as of this writing. It describes services everything the NetScaler can do, which is overkill for most tools. Fetching the default `NSConfig.wsdl` will cause `netscaler.py` to compile them all.

This can take a long time:

```
% time ./nstest.py
WSDL: file:///home/j/jathan/sandbox/NSConfig.wsdl
Starting client...
Done.
./netscaler.py  12.23s user 0.37s system 99% cpu 12.613 total
```

It will take even longer if you have to download the WSDL every time you start up your program. So you definitely want to filter your WSDL and the NetScaler has a CLI tool called `filterwsdl` that does just that.

If you want more details on why to do it, please read http://bit.ly/aX57SS.

So let's say we just want to interact with user administration operations. How about `login`, `logout`, `savensconfig` (of course), and anything with `systemuser` in it. It goes like this (run from CLI shell on NetScaler):

```
# filterwsdl /netscaler/api/NSConfig.wsdl +"log*" +"*systemuser*" +"savensconfig" > /netscaler/api/N
```

Then `scp` the file to localhost from the device. Now let's compare:

```
-rw-r--r--  1 jathan jathan 2.6M 2009-08-19 00:40 NSConfig.wsdl
-rw-r--r--  1 jathan jathan  14K 2010-03-02 16:36 NSUserAdmin.wsdl
```

Big difference. Observe how fast does subset WSDL compiles:

```
% time ./nstest.py
WSDL: file:///home/j/jathan/sandbox/NSUserAdmin.wsdl
Starting client...
Done.
./netscaler.py  0.36s user 0.03s system 100% cpu 0.392 total
```

HUGE difference.

CHAPTER

NINE

SUDS WSDL CACHING

Before we play with it there is one thing to keep in mind about `suds.client`. It will cache the WSDL by default, which is helpful for production but can be confusing while testing and debugging, especially if you're tweaking your filtered WSDL. So whenever testing, always pass `cache=None` to the constructor to avoid this confusion.

CHAPTER

TEN

COMMAND-LINE EXAMPLE

Ok now let's play with it:

```
>>> import netscaler
>>> wsdl_url = 'file:///Users/jathan/sandbox/netscaler-api/NSUserAdmin.wsdl'
>>> api = netscaler.API('netscaler', username='nsroot', password='nsroot', wsdl_url=wsdl_url, cache=N
setting username to nsroot
setting cache to None
setting password to nsroot
wsdl_url: file:///Users/jathan/sandbox/netscaler-api/NSUserAdmin.wsdl
soap_url: http://netscaler/soap/
```

Now if you print the api object, it acts just like a `suds.client.Client` object. Notice this subset of methods is way lower than the 2800+ methods from the master WSDL:

```
>>> print api

Suds ( https://fedorahosted.org/suds/ )  version: 0.3.9 GA  build: R659-20100219

Service ( NSConfigService ) tns="urn:NSConfig"
Prefixes (2)
    ns0 = "http://schemas.xmlsoap.org/soap/encoding/"
    ns1 = "urn:NSConfig"
Ports (1):
    (NSConfigPort)
        Methods (10):
            addsystemuser(xs:string username, xs:string password, )
            bindsystemuser_policy(xs:string username, xs:string policyname, xs:unsignedInt priority,
            getsystemuser(xs:string username, )
            login(xs:string username, xs:string password, )
            loginchallengeresponse(xs:string response, )
            logout()
            rmsystemuser(xs:string username, )
            savensconfig()
            setsystemuser_password(xs:string username, xs:string password, )
            unbindsystemuser_policy(xs:string username, xs:string policyname, )
        Types (54):
            ns0:Array
            ns0:ENTITIES
            ns0:ENTITY
            ns0:ID
            ns0:IDREF
            ns0:IDREFS
            ns0:NCName
            ns0:NMTOKEN
            ns0:NMTOKENS
            ns0:NOTATION
            ns0:Name
            ns0:QName
            ns0:Struct
            ns0:anyURI
            ns0:arrayCoordinate
            ns0:base64
            ns0:base64Binary
            ns0:boolean
            ns0:byte
            ns0:date
            ns0:dateTime
            ns0:decimal
            ns0:double
            ns0:duration
            ns0:float
            ns0:gDay
            ns0:gMonth
            ns0:gMonthDay
            ns0:gYear
            ns0:gYearMonth
            getsystemuserResult
            ns0:hexBinary
            ns0:int
            ns0:integer
            ns0:language
            ns0:long
            ns0:negativeInteger
            ns0:nonNegativeInteger
            ns0:nonPositiveInteger
            ns0:normalizedString
            ns0:positiveInteger
            ns0:short
            simpleResult
            ns0:string
            stringList
            systemuser
            systemuserList
            ns0:time
            ns0:token
            ns0:unsignedByte
            ns0:unsignedInt
            unsignedIntList
            ns0:unsignedLong
            ns0:unsignedShort
```

Now we can run a command:

```
>>> api.run("addsystemuser", username='jathan', password='jathan')
config changed, autosaving.
Done
(simpleResult){
    rc = 0
    message = "Done"
}
```

CHAPTER

ELEVEN

AUTOSAVE

Config changed, autosaving!

You might as yourself why not just directly invoke `api.client.service.addsystemuser()`. That's a good question. It depends on whether you want to take advantage of the little perks I added like automatic login and automatic saving of the configuration on volatile operations. Some people might like these ideas, others might not. Autosave is enabled by default, but you can disabled it by passing `autosave=False` to the constructor.

Currently any command that does not start with `login`, `logout`, `get`, or `save` is considered volatile, and will trigger an autosave.

CHAPTER

TWELVE

USERADMIN - A SUBCLASSING EXAMPLE

In the examples directory is `nsuser.py`, which is an example of how one might utilize subclassing to wrap some business logic around certain commands. Here it is:

```
class IllegalName(netscaler.InteractionError): pass

class UserAdmin(netscaler.API):
    def is_safe(self, username):
        """Returns False for names containing 'root' or starting with 'ns'."""
        if 'root' in username or username.startswith('ns'):
            return False
        return True

    def add_user(self, username, password):
        """Custom user adder that won't allow unsafe names"""
        if not self.is_safe(username):
            raise IllegalName(username)

        try:
            resp = self.run("addsystemuser", username=username, password=password)
            return True
        except netscaler.InteractionError, err:
            return False

    def del_user(self, username):
        """Custom user remover that protects usernames"""
        if not self.is_safe(username):
            raise IllegalName(username)

        try:
            resp = self.run("rmsystemuser", username=username)
            return True
        except netscaler.InteractionError, err:
            return False

    def user_exists(self, username):
        """Returns True if user exists."""
        try:
            resp = self.run("getsystemuser", username=username)
            return True
        except netscaler.InteractionError, err:
            return False
```

I used the example of blacklisting the creation or removal of any user that has "root" in the name or begins with "ns". So if you try any volatile operations on this user using this module, this is what happens:

```
>>> import nsuser
>>> wsdl_url = 'file:///Users/jathan/sandbox/netscaler-api/examples/NSUserAdmin.wsdl'
>>> api = nsuser.UserAdmin('netscaler', username='nsroot', password='nsroot',wsdl_url=wsdl_url, cache
>>> api.del_user('nsroot')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "nsuser.py", line 29, in del_user
    raise IllegalName(username)
nsuser.IllegalName: nsroot
```

If you run nsuser it does a little addition of missing users or removal of existing ones with some dummy accounts just to show how it works:

```
% py nsuser.py
setting username to nsroot
setting cache to None
setting password to nsroot
wsdl_url: file:///Users/jathan/sandbox/netscaler-api/examples/NSUserAdmin.wsdl
soap_url: http://netscaler/soap/
Done
logged in: True
autosave?  True

checking jathan
config changed; consider saving!
config changed; autosaving.
Done
jathan added!

checking dynasty
config changed; consider saving!
config changed; autosaving.
Done
dynasty added!

checking john
config changed; consider saving!
config changed; autosaving.
Done
john added!
```

And the other way:

```
% py nsuser.py
setting username to nsroot
setting cache to None
setting password to nsroot
wsdl_url: file:///Users/jathan/sandbox/netscaler-api/examples/NSUserAdmin.wsdl
soap_url: http://netscaler/soap/
Done
logged in: True
autosave?  True

checking jathan
jathan exists.
deleting
```

```
config changed; consider saving!
config changed; autosaving.
Done

checking dynasty
config changed; autosaving.
Done
dynasty exists.
deleting
config changed; consider saving!
config changed; autosaving.
Done

checking john
config changed; autosaving.
Done
john exists.
deleting
config changed; consider saving!
config changed; autosaving.
Done
```

END TRANSMISSION

CHAPTER

THIRTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*