dataflake.ldapconnection Documentation

Release 2.2.dev0

Jens Vagelpohl

Jun 29, 2018

Contents

`dataflake.ldapconnection` provides an abstraction layer on top of pyldap. It offers a connection object with simplified methods for inserting, modifying, searching and deleting records in the LDAP directory tree. Failover/redundancy can be achieved by supplying connection data for more than one LDAP server.

Narrative documentation

Narrative documentation explaining how to use `dataflake.ldapconnection`.

1.1 Installation

You will need [Python](#) version 2.7 or better to run `dataflake.ldapconnection`.

It is advisable to install `dataflake.ldapconnection` into a virtualenv in order to obtain isolation from any "system" packages you've got installed in your Python version (and likewise, to prevent `dataflake.ldapconnection` from globally installing versions of packages that are not compatible with your system Python).

After you've got the requisite dependencies installed, you may install `dataflake.ldapconnection` into your Python environment using the following command:

```
$ easy_install dataflake.ldapconnection
```

or:

```
$ pip install dataflake.fakeldap
```

If you use `zc.buildout` you can add `dataflake.fakeldap` to the necessary `eggs` section to have it pulled in automatically.

When you `easy_install` or `pip dataflake.fakeldap`, the pyldap libraries are installed if they are not present.

1.2 Using dataflake.ldapconnection

`dataflake.ldapconnection` provides an abstraction layer on top of pyldap. It offers a connection object with simplified methods for inserting, modifying, searching and deleting records in the LDAP directory tree. Failover/redundancy can be achieved by supplying connection data for more than one LDAP server.

1.2.1 API examples

Instantiating a connection object:

```
1  >>> from dataflake.ldapconnection.connection import LDAPConnection
2  >>> conn = LDAPConnection()
3  >>> conn.addServer('localhost', '1389', 'ldap')
```

To work with the connection object you need to make sure that a LDAP server is available on the provided host and port.

Now we will search for a record that does not yet exist, then add the missing record and find it when searching again:

```
1   >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
2   {'exception': '', 'results': [], 'size': 0}
3   >>> data = { 'objectClass': ['top', 'inetOrgPerson']
4   ...          , 'cn': 'testing'
5   ...          , 'sn': 'Lastname'
6   ...          , 'givenName': 'Firstname'
7   ...          , 'mail': 'test@test.com'
8   ...          , 'userPassword': '5ecret'
9   ...          }
10  >>> conn.insert('ou=users,dc=localhost', 'cn=testing', attrs=data, bind_dn=
    ↪'cn=Manager,dc=localhost', bind_pwd='secret')
11  >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
12  {'exception': '', 'results': [{'dn': 'cn=testing,ou=users,dc=localhost', 'cn': [
    ↪'testing'], 'objectClass': ['top', 'inetOrgPerson'], 'userPassword': ['5ecret'], 'sn
    ↪': ['Lastname'], 'mail': ['test@test.com'], 'givenName': ['Firstname']}], 'size': 1}
```

We can edit an existing record:

```
1  >>> changes = {'givenName': 'John', 'sn': 'Doe'}
2  >>> conn.modify('cn=testing,ou=users,dc=localhost', attrs=changes, bind_dn=
   ↪'cn=Manager,dc=localhost', bind_pwd='secret')
3  >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
4  {'exception': '', 'results': [{'dn': 'cn=testing,ou=users,dc=localhost', 'cn': [
   ↪'testing'], 'objectClass': ['top', 'inetOrgPerson'], 'userPassword': ['5ecret'], 'sn
   ↪': ['Doe'], 'mail': ['test@test.com'], 'givenName': ['John']}], 'size': 1}
```

As the last step, we will delete our testing record:

```
1  >>> conn.delete('cn=testing,ou=users,dc=localhost', bind_dn='cn=Manager,dc=localhost',
   ↪ bind_pwd='secret')
2  >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
3  {'exception': '', 'results': [], 'size': 0}
```

The *Interfaces* page contains more information about the connection APIs.

1.2.2 Handling string encoding for input and output values

LDAP servers expect values sent to them in specific string encodings. Standards-compliant LDAP servers use UTF-8. They use the same encoding for values returned e.g. by a search. This server-side encoding may not be convenient for communicating with the dataflake.ldapconnection API itself. For this reason the server-side encoding and API encoding can be set individually on connection instances using the attributes ldap_encoding and api_encoding, respectively. The connection instance handles all string encoding transparently.

By default, instances use UTF-8 as `ldap_encoding` and ISO-8859-1 (Latin-1) as `api_encoding`. You can assign any valid Python codec name to these attributes. Assigning an empty value or None means that unencoded unicode strings are used under Python 2.

1.3 Development

1.3.1 Getting the source code

The source code is maintained in the Dataflake Git repository. To check out the trunk:

```
$ git clone https://github.com/dataflake/dataflake.ldapconnection.git
```

You can also browse the code online at https://github.com/dataflake/dataflake.ldapconnection

1.3.2 Bug tracker

For bug reports, suggestions or questions please use the GitHub issue tracker at https://github.com/dataflake/dataflake.ldapconnection/issues.

1.3.3 Running the tests in a `virtualenv`

If you use the `virtualenv` package to create lightweight Python development environments, you can run the tests using nothing more than the `python` binary in a virtualenv. First, create a scratch environment:

```
$ /path/to/virtualenv --no-site-packages /tmp/virtualpy
```

Next, get this package registered as a "development egg" in the environment:

```
$ /tmp/virtualpy/bin/python setup.py develop
```

Finally, run the tests using the build-in `setuptools` testrunner:

```
$ /tmp/virtualpy/bin/python setup.py test
running test
...
test_escape_dn (dataflake.ldapconnection.tests.test_utils.UtilsTest) ... ok


----------------------------------------------------------------------
Ran 88 tests in 0.058s

OK
```

If you have the `nose` package installed in the virtualenv, you can use its testrunner too:

```
$ /tmp/virtualpy/bin/easy_install nose
...
$ /tmp/virtualpy/bin/python setup.py nosetests
running nosetests
......................................................................
.............................
----------------------------------------------------------------------
Ran 101 tests in 0.162s
```

```
OK
```

or:

```
$ /tmp/virtualpy/bin/nosetests
.......................................................................
...............................
----------------------------------------------------------------------
Ran 101 tests in 0.160s

OK
```

If you have the `coverage` package installed in the virtualenv, you can see how well the tests cover the code:

```
$ /tmp/virtualpy/bin/easy_install nose coverage
...
$ /tmp/virtualpy/bin/python setup.py nosetests \
    --with-coverage --cover-package=dataflake.ldapconnection
running nosetests
...

Name                                  Stmts   Exec  Cover   Missing
------------------------------------------------------------------
dataflake.ldapconnection                  1      1   100%
dataflake.ldapconnection.connection     246    244    99%   214-215
dataflake.ldapconnection.interfaces      10     10   100%
dataflake.ldapconnection.utils            7      7   100%
------------------------------------------------------------------
TOTAL                                   264    262    99%
-------------------------------------------------------------------
Ran 101 tests in 0.226s

OK
```

1.3.4 Running the tests using `zc.buildout`

`dataflake.ldapconnection` ships with its own `buildout.cfg` file and `bootstrap.py` for setting up a development buildout:

```
$ python bootstrap.py
...
Generated script '.../bin/buildout'
$ bin/buildout
...
```

Once you have a buildout, the tests can be run as follows:

```
$ bin/test --all
Running tests at all levels
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Running:
.......................................................................
........................
```

```
  Ran 94 tests with 0 failures and 0 errors in 0.042 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
```

1.3.5 Building the documentation using `zc.buildout`

The `dataflake.ldapconnection` buildout installs the Sphinx scripts required to build the documentation, including testing its code snippets:

```
$ cd docs
$ make doctest
Running Sphinx v1.6.5
...
running tests...


Doctest summary
===============
    0 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests sn the sources finished, look at the  results in \
    .../docs/_build/doctest/output.txt.
```

1.3.6 Making a release

These instructions assume that you have a development sandbox set up using `zc.buildout` as the scripts used here are generated by the buildout.

```
$ bin/buildout -o
$ python setup.py sdist bdist_wheel upload --sign
```

The `bin/buildout` step will make sure the correct package information is used.

1.4 Change log

1.4.1 2.2 (unreleased)

1.4.2 2.1 (2018-06-29)

- test and declare support for Python 3.7

1.4.3 2.0 (2017-10-29)

- Python 3 compatibility
- Move from `python-ldap` to `pyldap` for Python 3 compatibility
- PEP 8 code cleanup

- added Travis CI automatic testing configuration

- use pkgutil-style namespace declaration

- package cleanup (`.gitignore`, `MANIFEST.in`, `README.rst`)

- docs cleanup (`Makefile`, `conf.py`)

- tests cleanup (`tox.ini`, `.travis.yml`)

- remove unsupported documentation bits

- fix coverage tests to only test this package

- remove coveralls from the Travis CI configuration

- make the character sets used for communicating with the LDAP libraries (`ldap_encoding`) and the calling side (`api_encoding`) configurable in the `LDAPConnection` constructor.

1.4.4 1.5 (2012-04-20)

- Factored the `dataflake.ldapconnection.tests.fakeldap` module into a separate package `dataflake.fakeldap`

- Restricted the supported Python versions to 2.6 and 2.7.

- removed old test_suite fossils

1.4.5 1.4 (2012-04-03)

- Added a simple tox configuration to quickly test with different Python versions. As a result of successful testing, re-declare compatibility with Python 2.4 and 2.5.

- Moved the documentation build dependencies into a setup extra so building them without buildout becomes easier.

- Added setup.py aliases for creating a testing sandbox and the documentation dependencies, and use them in the buildout configuration.

- Extended `fakeldap.RaisingFakeLDAPConnection` to accept a list of exceptions to raise. On each call to the method that is set to raise the exception, the first item in the exception list is removed and raised. This allows testing code areas nested in more than one `try/except` clause.

1.4.6 1.3 (2012-03-23)

- Update bootstrap.py to what ships with zc.buildout 1.5.2

- fakeldap: Add email characters and some non-ASCII characters to FLTR_RE, to be able to use them in searches. (Stefan Holek)

- fakeldap: Add unbind_s API. (Stefan Holek)

- fakeldap: Deep-copy entries before returning them. (Stefan Holek)

- fakeldap: Only return requested attributes. (Stefan Holek)

- fakeldap: Optionally disable password hashing. (Stefan Holek)

- fakeldap: Optionally maintain the 'memberOf' attribute of group members. (Stefan Holek)

- connection: Clean up encoding and decoding of values for when the api_encoding is set to None. (Stefan Holek)

- connection: New 'raw' kwarg for the search API. If true, search results are returned in the ldap_encoding. (Stefan Holek)

- connection: Store a connection's bind_dn and bind_pwd as is, and encode them before use. (Stefan Holek)

- connection: Fix a condition that caused rebinding to fail if only one of bind_dn and bind_pwd had changed. (Stefan Holek)

- connection: Add disconnect API. (Stefan Holek)

- connection: Allow to delete subsets from multi-valued attributes. (Stefan Holek)

- switched documentation to point to the new Git repository

1.4.7 1.2 (2010-08-09)

- Using `id()` is not random enough for a unique hash.

1.4.8 1.1 (2010-05-09)

- Updated Sphinx Makefile and configuration to be closer to the latest Sphinx version

- Greatly expand installation and testing documentation using ideas from Tres Seaver

1.4.9 1.0 (2010-04-12)

- Bug: fakeldap.FakeLDAPConnection wildcard searches did not work correctly and returned too many matches.

- Bug: Improve behavior matching of standard python-ldap and fakeldap by raising ldap.NO_SUCH_OBJECT where operations target non-existing entries.

- Bug: Improve behavior matching of standard python-ldap and fakeldap by raising ldap.ALREADY_EXISTS where operations duplicate existing entries.

- Bug: Added tests for all fakeldap.FakeLDAPConnection methods and added tests for some other module classes and functions.

- Refactoring: Removed the fakeldap.initialize and explode_dn functions. They were either not needed or needlessly duplicating existing python-ldap features.

- Bug: python-ldap will no longer support the LDAP connection class ldap.ldapobject.SmartLDAPObject with version 2.3.11. Replacing it with ReconnectLDAPObject.

- Bug: If a connection raised an LDAP exception inside start_tls_s handling was broken.

- Feature: You can now add server definitions for servers that support the StartTLS extended operation. Whereas the existing secure connections using the `ldaps` protocol are encrypted throughout, StartTLS is used through an unencrypted connection to request all further traffic to be encrypted.

- Refactoring: Switch tests to using the fakeldap LDAP connection object wherever possible, and correct a few fakeldap and LDAPConnection misbehaviors along the way.

1.4.10 1.0b1 (2010-02-01)

- Performing more rigorous input checking for DNs

- Made encoding/decoding more flexible by adding configuration flags for the encoding used by the LDAP server and the encoding for calls to and return values from the connection API. The default is backwards compatible (UTF-8 for the LDAP server encoding, and Latin-15 for the API encoding).

- Factored the connection tests module into a series of modules, it was getting large and unwieldy.

- move the actual python-ldap connection from an attribute into a module-level cache since those connections cannot be pickled.

- Removed the rdn_attr attribute, which was used to try and determine if a modify operation should trigger a modrdn. We now fish the RDN attribute from the record's DN for this purpose.

- Changed the way internal logging is done to avoid storing logger objects onto the connection instance unless it is explicitly specified. This means the instance is picklable when using the default logging.

- Removed the *bind* method. There was no good reason to expose it as part of the public API, and since bind operations are re-done as part of all operations it would only serve to confuse users. Users who want to use credentials other than the credentials configured into the connection instance should pass them along explicitly when invoking the operation.

- The search method now provides a default search subtree search scope if none is specified.

- Creating a new instance does not require passing server data like host, port and protocol anymore.

- replaced several methods with better alternatives from python-ldap, which also requires upping the dependency to python-ldap>=2.3.0, and fixing up the tests.

- pare down fakeldap to not try and provide all kinds of constants from python-ldap, but just a LDAP connection class.

- add a new method "bind" to rebind a connection, if the last bind differs from the desired bind.

- rename variable name "filter" with "fltr" to stop shadowing the Python function "filter".

- added an interfaces file as documentation and "contract". This adds a dependency on zope.interface.

- removed unused argument "login_attr" from constructor argument list

- LDAPConnection objects now accept more than a single server definition. Failover between connections is triggered by connection or operation timeouts. Added API to add and remove server definitions at runtime.

- all those methods causing LDAP operations to be performed accept optional bind_dn and bind_pwd named arguments to rebind with the provided credentials instead of those credentials stored in the LDAPConnection instance. This represents an API change for the *insert*, *modify* and *delete* methods.

1.4.11 0.4 (2008-12-25)

- fakeldap bug: the modify_s method would expect changes of type MOD_DELETE to come with a list of specific attribute values to delete. Now the attribute will be deleted as a whole if the expected list is None, this reflects actual python-ldap behavior better.

- now we are exercising the fakeldap doctests from within this package, they used to be run from Products.LDAPUserFolder, which was not cleaned up when the fakeldap module moved to dataflake.ldapconnection.

1.4.12 0.3 (2008-08-30)

- fakeldap: no longer override the LDAP exceptions, just get them from python-ldap. (http://www.dataflake.org/tracker/issue_00620)

1.4.13 0.2 (2008-08-27)

- backport a fix applied to the LDAPUserFolder FakeLDAP module to handle BASE-scoped searches on a DN.

1.4.14 0.1 (2008-06-11)

- Initial release.

API documentation

API documentation for `dataflake.ldapconnection`.

2.1 Interfaces

Note: On ReadTheDocs this page will not work correctly.

2.2 `dataflake.ldapconnection.connection`

Note: On ReadTheDocs this page will not work correctly.

Support

If you need commercial support for this software package, please see https://www.zetwork.com.

Indices and tables

- genindex
- modindex
- search
- glossary

Python Module Index

d

Index

D