LDAP and the Python landscape

Ondřej Kuzník

Symas

WhoAmI

```
dn: cn=Ondřej Kuzník,ou=people
worksOn: OpenLDAP
worksOn: python-ldap
worksOn: TUIs
worksOn: distributed systems
worksOn: automating things that should not be automated
```

Why this talk

CRUD is "easy"

- and then you realise you're not the only application touching the directory

"I just need a Bind, right?"

- and account status
- and group membership
- and ...

python-ldap

Started in 1998, latest release 3.4.8 (2026)

Native module wrapping libldap plus abstraction code in Python

API shape:

- very close to libldap
- open a connection, issue operations, optionally collect results
- a non-success result raises an exception

Extras:

- ldif, url parsing and generation
- `slapdtest` module

ldap3

Started in ~2013, latest release 2.9.1 (2021), 2.10.2rc4 (2026)

Pure Python LDAP implementation

API shape:

- define `Server`, open `Connection`s and choose a `strategy`

Extras:

- connection pooling
- ldif

bonsai

Started in 2015, latest release 1.5.5 (2026)

Native module wrapping libldap on Linux, winldap on Windows

API shape:

- middle ground, open a connection, issue operations,

Extras:

- connection pool
- ldif

The feel

A simple task: Connect to an LDAP server, set up TLS, bind and get some data back.

simple bind+search

python-ldap

```
l = LDAPObject("ldap://ldap.example.com")
l.set_option(ldap.OPT_X_TLS_REQUIRE_CERT, ldap.OPT_X_TLS_DEMAND)
l.set_option(ldap.OPT_X_TLS_NEWCTX, 0)
l.start_tls_s()
msgid = l.simple_bind("cn=admin,dc=example,dc=com", "secret")
l.result(msgid)  # or l.result4

entries = l.search_ext_s("", ldap.SCOPE_BASE, attrlist=["+"])
dn, attrs = entries[0]
```

ldap3

```
server = Server("ldap.example.com", use_ssl=True,
                tls=Tls(validate=ssl.CERT_REQUIRED))
conn = Connection(server, "cn=admin,dc=example,dc=com", "secret",
                  client_strategy=SAFE_SYNC, auto_bind=True)
status, result, response, _ = conn.search(
    "", "", BASE, attributes=["+"])

entries = [r for r in response if r["type"] == "searchResEntry"]
entry = entries[0]
entry.entry_dn, entry.attr or entry['attr']
```

bonsai

```
client = LDAPClient("ldap://ldap.example.com", tls=True)
client.set_cert_policy("demand")
client.set_credentials("SIMPLE", user="cn=admin,dc=example,dc=com",
                       password="secret")
with client.connect(is_async=True) as conn:
    entries = await conn.search("", LDAPSearchScope.BASE, "",
                                attrList=["+"])
    entry = entries[0]
    entry.dn, entry['attr']
```

Advanced usage

LDAP Schema

python-ldap

Use `ldap.schema`.

ldap3

Supports loading schema from server and stored in library, enabled by default.

bonsai

No first class schema support in library

ORM-like features

python-ldap

- Not in library

ldap3

```
c.search(...)
e = c.entries[0].entry_writable()
e.givenname = "John"
e.cn += "John Doe"
e.entry_commit_changes()
```

bonsai

```
result = conn.search(...)
entry = search[0]
entry['mail'] = "user@example.com"
entry.modify()
```

Pipelining

Not in bonsai.

python-ldap

```
conn = Connection(...,  client_strategy=ASYNC)
sent = [l.search_ext(dn, ldap.SCOPE_BASE)
        for dn in (ALICE, BOB, CAROL)]
while pending:
    rtype, _, msgid, _, _, _ = l.result4(
        ldap.RES_ANY, all=0, timeout=5)
```

ldap3

```
ids = [conn.search(dn, "(objectClass=*)")
       for dn in (ALICE, BOB, CAROL)]
for msgid in ids:
    response, result = conn.get_response(msgid)
```

Extensions

Controls

Control	python-ldap	ldap3	bonsai
Paging	✓	✓	✓
SSS/VLV	✓	✓	✓
ManageDSAiT	✓	✓	✓
pre/post read	✓	✓	✓
assert	✓	must encode	✗
ppolicy	✓	must decode	✓
Proxy Authz	✓	must encode	✗
Session Tracking	✓	must encode	✗
Arbitrary	full	must-(en/de)code	✗

Extensions

Exop	python-ldap	ldap3	bonsai
Increment	✓	✓	✗
WhoAmI	✓	✓	✓
Password Modify	✓	✓	✓
Arbitrary	full	must-decode	✗
Intermediates	✓	~	✗
Open-ended requests	✓	ASYNC_* strategies	✗

Miscellaneous

Asyncio

python-ldap

Not exposed by default, simple wrappers exist, fully async iterable (searches, intermediate results, ...).

ldap3

No asyncio strategy, `ASYNC` strategy spawns a background thread

bonsai

Just `connect(is_async=True)` and the resulting object's operations are awaitable. Several implementations supported (asyncio, trio, ...) No in-progress data.

Performance

- using libldap for I/O is a clear performance win, matters for higher bandwidth applications

Use case:

- pyfortress migration `ldap3` → `python-ldap`

future

- python-ldap major improvements in API usability

```
conn = Connection("ldap://ldap.example.com")
conn.set_option(ldap.OPT_X_TLS_REQUIRE_CERT, ldap.OPT_X_TLS_DEMAND)
conn.set_option(ldap.OPT_X_TLS_NEWCTX, 0)
request = conn.start_tls()
result = conn.result(request)
result.raise_for_result()

request = conn.simple_bind("cn=admin,dc=example,dc=com", "secret", attrList=["+"])
request.result().raise_for_result()

request = conn.search("", ldap.SCOPE_BASE, "", attrs=["+"])
for msg in conn.result(request):
    match msg:
        case SearchEntry() as entry:
            entry.dn, entry.attrs
        case SearchResult() as result:
            result.raise_for_result()
```

Connection pooling

Connection pools are tricky, do not hide session management from code

- e.g. ldap3 with the pooling strategy pools internally
- bonsai lets you borrow a connection from it, return when done

Some controls change the session, the former is not always safe.

Servers and testing

slapdtest

- stand up a real OpenLDAP server
- tools to configure and populate it
- start/restart stop capabilities

ldap3 MOCK_SYNC strategy

```
conn = Connection(..., client_strategy=MOCK_SYNC)
conn.strategy.entries_from_json('my_entries.json')
# or
conn.strategy.add_entry('cn=user0,o=lab', {'userPassword': ..., ...})
...
```

pldap.server

Build your own server

```
class Server(pldap.server.Server):
    async def search(self, request):
        await asyncio.sleep(1)
        yield SearchResultEntry(
            msgid=request.msgid, dn="dc=example,dc=com",
            attributes={...}, controls=[])
        yield SearchResult(
            msgid=request.msgid, resultCode=0,
            matchedDN="", diagnosticMessage="",
            controls=None)
```