Understanding Plugin Architecture for Python Packages with SQLAlchemy Dialects

Ashish Shukla

Lead Software Engineer

EPAM Systems

[epam_logo]

Overview

Plugins in context of SQLAlchemy

- A look at SQL Alchemy dialects and exploring create_engine
- Python package entrypoints and importlib.metadata

How you can make it work for your package

- Creating a hello world package
- Creating plugin structure and plugin loader utils
- Extending hello world package with hello-world-kannada-plugin

Plugins in context of SQLAlchemy

A new web framework comes every other month in Python but SQLALchemy is still there. *despite not so great documentation*

The reason is - its extensibility.

Good software needs to be extensible.

Plugins in context of SQLAlchemy

Dialects

From SQLAlchemy docs

> the “dialect” is a Python object that represents information and methods that allow database operations to proceed on a particular kind of database backend and a particular kind of Python driver (or DBAPI) for that database.

Examples - Postgresql is a dialect.

Find class definition of dialect here

Plugins in context of SQLAlchemy

Dialects (contd..)

SQLAlchemy gives a few dialects for a few databases (mssql, mysql, oracle, postgresql, sqlite)

You can find them at `lib/sqlalchemy/dialects`

- But there are many more supported !
  eg. Clickhouse, Druid, Drill, Snowflake, Impala, Google sheets and the list goes on

Plugins in context of SQLAlchemy

Create Engine

```python
from sqlalchemy import create_engine

engine = create_engine(
    "postgresql+psycopg2://user:pwd@localhost:5432/db"

)
```

The url scheme is `[dialect]+[driver]://...`

Plugins in context of SQLAlchemy

What is Create Engine doing ?

A lot.

But what we are interested in is this -

Plugins in context of SQLAlchemy

```python
def create_engine(url: str, **kwargs):
    u = _url.make_url(url) # _url is the url module that does url magic

    ...

    entrypoint = u._get_entrypoint() # Gets the dialect's entrypoint class
    dialect_cls = entrypoint.get_dialect_cls(u) # Gets the actual dialect class
    dialect_args = _create_dialect_args(url, **kwargs) # transforms kwargs to dialect specific args

    # Intense Python wizardry ...

    dialect = dialect_cls(**dialect_args)
    engine = engineclass(poo, dialect, u, **kwargs)

    # Some more Spells ...
    return engine
```

Note: Code lines cherry picked for sanity ! The full thing is here.

Plugins in context of SQLAlchemy

A little deeper -

```
def _get_entrypoint():
    ...
    cls = registry.load(name)
    ...
    return cls
```

The registry is where all of the dialects are held.

The registry is declared like this -

```
registry = util.PluginLoader("sqlalchemy.dialects")
```

Plugins in context of SQLAlchemy

Last stretch! Now's the time to focus !!

```python
from importlib import metadata as importlib_metadata

class PluginLoader:
    def __init__(self, group):
        self.group = group
        self.impls = {}

    def load(self, name):
        if name in self.impls:
            return self.impls[name]()

        entrypoints = importlib_metadata.entry_points()
        impls = entrypoints.select(self.group)
        for impl in impls:
            if impl.name == name:
                self.impls[impl.name] = impl.load
                return impl.load()

        raise ValueError("No such plugin!")
```

How you can make it work for your package

Using the same logic for our own plugins

1. Let others know how they can create code for you
2. Give them an identifier so that they can tell their code is for your package
3. Write some code that can load `their code`
4. Use others' code !

How you can make it work for your package

Coding Time !!

Thanks !

Question ?

Code and slides at:

https://github.com/ash2shukla/pycon2025

Connect with me:

https://linkedin.com/in/ash2shukla