2.2. Connecting to MongoDB

Connections in MongoEngine are registered globally and are identified with aliases. If no `alias` is provided during the connection, it will use “default” as alias.

To connect to a running instance of **mongod**, use the `connect()` function. The first argument is the name of the database to connect to:

```
from mongoengine import connect
connect('project1')
```

By default, MongoEngine assumes that the **mongod** instance is running on **localhost** on port **27017**.

If MongoDB is running elsewhere, you need to provide details on how to connect. There are two ways of doing this. Using a connection string in URI format (**this is the preferred method**) or individual attributes provided as keyword arguments.

2.2.1. Connect with URI string

When using a connection string in URI format you should specify the connection details as the `host` to `connect()`. In a web application context for instance, the URI is typically read from the config file:

```
connect(host="mongodb://127.0.0.1:27017/my_db")
```

If the database requires authentication, you can specify it in the URI. As each database can have its own users configured, you need to tell MongoDB where to look for the user you are working with, that’s what the `?authSource=admin` bit of the MongoDB connection string is for:

```
# Connects to 'my_db' database by authenticating
# with given credentials against the 'admin' database (by default as authSource isn't
provided)
connect(host="mongodb://my_user:my_password@127.0.0.1:27017/my_db")

# Equivalent to previous connection but explicitly states that
# it should use admin as the authentication source database
connect(host="mongodb://my_user:my_password@hostname:port/my_db?authSource=admin")

# Connects to 'my_db' database by authenticating
# with given credentials against that same database
connect(host="mongodb://my_user:my_password@127.0.0.1:27017/my_db?authSource=my_db")
```

The URI string can also be used to configure advanced parameters like ssl, replicaSet, etc. For more information or example about URI string, you can refer to the official doc:

```
connect(host="mongodb://my_user:my_password@127.0.0.1:27017/my_db?
authSource=admin&ssl=true&replicaSet=globaldb")
```

Note

URI containing SRV records (e.g “mongodb+srv://server.example.com/”) can be used as well

2.2.2. Connect with keyword attributes

The second option for specifying the connection details is to provide the information as keyword attributes to `connect()`:

```
connect('my_db', host='127.0.0.1', port=27017)
```

If the database requires authentication, `username`, `password` and `authentication_source` arguments should be provided:

```
connect('my_db', username='my_user', password='my_password',
authentication_source='admin')
```

The set of attributes that `connect()` recognizes includes but is not limited to: `host`, `port`, `read_preference`, `username`, `password`, `authentication_source`, `authentication_mechanism`, `replicaset`, `tls`, etc. Most of the parameters accepted by pymongo.MongoClient can be used with `connect()` and will simply be forwarded when instantiating the *pymongo.MongoClient*.

Note

Database, username and password from URI string overrides corresponding parameters in `connect()`, this should obviously be avoided:

```
connect(
    db='test',
    username='user',
    password='12345',
    host='mongodb://admin:qwerty@localhost/production'
)
```

will establish connection to `production` database using `admin` username and `qwerty` password.

Note

Calling `connect()` without argument will establish a connection to the “test” database by default

2.2.3. Read Preferences

As stated above, Read preferences are supported through the connection but also via individual queries by passing the read_preference

```
from pymongo import ReadPreference

Bar.objects().read_preference(ReadPreference.PRIMARY)
Bar.objects(read_preference=ReadPreference.PRIMARY)
```

2.2.4. Multiple Databases

To use multiple databases you can use `connect()` and provide an *alias* name for the connection - if no *alias* is provided then “default” is used.

In the background this uses `register_connection()` to store the data and you can register all aliases up front if required.

2.2.4.1. Documents defined in different database

Individual documents can be attached to different databases by providing a *db_alias* in their meta data. This allows `DBRef` objects to point across databases and collections. Below is an example schema, using 3 different databases to store data:

```
connect(alias='user-db-alias', db='user-db')
connect(alias='book-db-alias', db='book-db')
connect(alias='users-books-db-alias', db='users-books-db')

class User(Document):
    name = StringField()

    meta = {'db_alias': 'user-db-alias'}

class Book(Document):
    name = StringField()

    meta = {'db_alias': 'book-db-alias'}

class AuthorBooks(Document):
    author = ReferenceField(User)
    book = ReferenceField(Book)

    meta = {'db_alias': 'users-books-db-alias'}
```

2.2.4.2. Disconnecting an existing connection

The function `disconnect()` can be used to disconnect a particular connection. This can be used to change a connection globally:

```
from mongoengine import connect, disconnect
connect('a_db', alias='db1')

class User(Document):
    name = StringField()
    meta = {'db_alias': 'db1'}

disconnect(alias='db1')

connect('another_db', alias='db1')
```

Note

Calling `disconnect()` without argument will disconnect the “default” connection

Note

Since connections gets registered globally, it is important to use the *disconnect* function from MongoEngine and not the *disconnect()* method of an existing connection (pymongo.MongoClient)

Note

`Document` are caching the pymongo collection. using *disconnect* ensures that it gets cleaned as well

2.2.5. Context Managers

Sometimes you may want to switch the database or collection to query against. For example, archiving older data into a separate database for performance reasons or writing functions that dynamically choose collections to write a document to.

2.2.5.1. Switch Database

The `switch_db` context manager allows you to change the database alias for a given class allowing quick and easy access to the same User document across databases:

```
from mongoengine.context_managers import switch_db

class User(Document):
    name = StringField()

    meta = {'db_alias': 'user-db'}

with switch_db(User, 'archive-user-db') as User:
    User(name='Ross').save()  # Saves the 'archive-user-db'
```

Note

`switch_db()` when used on a class that allow inheritance will change the database alias for instances of a given class only - instances of subclasses will still use the default database.

2.2.5.2. Switch Collection

The `switch_collection()` context manager allows you to change the collection for a given class allowing quick and easy access to the same Group document across collection:

```
from mongoengine.context_managers import switch_collection

class Group(Document):
    name = StringField()

Group(name='test').save()  # Saves in the default db

with switch_collection(Group, 'group2000') as Group:
    Group(name='hello Group 2000 collection!').save()  # Saves in group2000 collection
```

Note

Make sure any aliases have been registered with `register_connection()` or `connect()` before using the context manager.