Approzium Python SDK

Jan 13, 2022

User Guide

This is the Python SDK for Approzium (identity-based credential-less authentication to databases). Currently, there is support for Psycopg2 and Asyncpg (both for Postgres) as database drivers, but support for more database drivers is planned. Check out our roadmap for more details.

Approzium Python SDK is implemented as thin wrappers that integrate with existing Python database drivers, resulting in being extremely easy to use. It creates the same database connection objects that you are using, so you don't have to change your existing code!

Currently, it supports Python 3.5+ and AWS-based identity.

CHAPTER 1

Contents

1.1 Installation

The last stable release is available on PyPI and can be installed with `pip`:

```
$ pip3 install approzium
```

This installs only the approzium SDK. If you would like to additionally install all supported database drivers libraries, run:

```
$ pip3 install 'approzium[sqllibs]'
```

To install Opentelemetry and other dependencies for generating tracing:

```
$ pip3 install 'approzium[tracing]'
```

1.2 Requirements

- CPython >= 3.5

1.3 Supported Database Drivers

The following database driver libraries are supported:

Database	Driver	Authentication Methods	Notes
Postgres	Psycopg2	MD5* (Postgres default) and SCRAM-SHA-256 authentication	
Postgres	Asyncpg	Same as above	
MySQL	MySQL Connector	Native password authentication	Currently, only the pure Python implementation is supported
MySQL	PyMySQL	Same as above	

Warning: Even though MD5 is cryptographically insecure, it is the default authentication method used by Postgres, and so we made the decision to support it. However, we recommend using the more secure SCRAM-SHA256 authentication when possible.

1.4 Usage

Approzium Python SDK is designed to have a small footprint on the source code of your application.

1. The first step in creating an Approzium database connection is instantiating an *approzium.AuthClient*:

```
import approzium
auth = approzium.AuthClient("authenticator_service_host:port")
```

2. By default, the AuthClient automatically detects the environment that the service is running in. Currently, only AWS-based IAM identity is supported, so it will detect that.

3. Set this auth client to be the default one:

```
approzium.default_auth_client = auth
```

4. Create a connection! The way you create a connection is extremely similar to existing code. All you have to do is prepend `approzium.` to the import path. For example, if you are creating a Psycopg2 connection, instead of `psycopg2.connect` you would use `approzium.psycopg2.connect`. It's that easy!

```
from approzium.psycopg2 import connect
conn = connect(dbname="test", user="postgres", host="host.com")
# conn is now a psycopg2.Connection object
```

1.5 AuthClient

approzium.**default_auth_client = None**
Set this variable to an instance of *AuthClient* to set it as the default auth client to be used for connections.

class approzium.**AuthClient**(*server_address*, *disable_tls=False*, *tls_config=None*, *iam_role=None*)
Bases: object

This class represents a connection to an Approzium authenticator service. Instances of this class can be used as arguments to database drivers connect method to use for authentication.

Parameters

- **server_address** (*str*) – address (host:port) at which an authenticator service is listening.
- **disable_tls** (*bool, optional*) – defaults to False. When False, https is used and a client_cert and client_key proving the client's identity must be provided. When True, http is used and no other TLS options must be set.
- **tls_config** (TLSConfig, *optional*) – the TLS config to use for encrypted communication.
- **iam_role** (*str, optional*) – if an IAM role Amazon resource number (ARN) is provided, it will be assumed and its identity will be used for authentication. Otherwise, the default boto3 session will be used as the identity.

attribution_info

Provides a dictionary containing information about the current state of the AuthClient. Useful for logging.

Return type dict

Return Structure:

- *authenticator_address* (*str*): address of authenticator service used
- *iam_arn* (*str*): IAM Amazon resource number (ARN) used as identity
- *authenticated* (*bool*): whether the AuthClient was verified by the authenticator service.
- *num_connections* (*int*): number of connections made through this AuthClient

attribution_info_json

Provides the same attribution info returned by *attribution_info()* as a JSON format string

Return type str

class approzium.**TLSConfig**(*trusted_certs=None*, *client_cert=None*, *client_key=None*)

Bases: object

This class represents the TLS config to be used while communicating with Approzium. Its fields are further described here: https://grpc.github.io/grpc/python/grpc.html#create-client-credentials

Parameters

- **trusted_certs** (*str, optional*) – the path to the root certificate(s) that must have issued the identity certificate used by Approzium's authentication server.
- **client_cert** (*str, optional*) – this client's certificate, used for proving its identity, and used by the caller to encrypt communication with its public key
- **client_key** (*str, optional*) – this client's key, used for decrypting incoming communication that was encrypted by callers using the client_cert's public key

1.6 approzium.psycopg2

1.7 approzium.asyncpg

approzium.asyncpg.**connect**(**args*, *authenticator=None*, ***kwargs*)

Creates a Asyncpg connection through Approzium authentication. Takes the same arguments as asyncpg.connect, in addition to the authenticator argument.

Parameters **authenticator** (approzium.AuthClient, *optional*) – AuthClient instance to be used for authentication. If not provided, the default AuthClient, if set, is used.

Raises TypeError, if no AuthClient is given and no default one is set.

Return type asyncpg.Connection

Example:

```
>>> import approzium
>>> import asyncio
>>> from approzium.asyncpg import connect
>>> auth = approzium.AuthClient("myauthenticator.com:6001", disable_tls=True)
>>> async def run():
...     con = await connect(user='postgres', authenticator=auth)
...     # use the connection just like any other Asyncpg connection
...     types = await con.fetch('SELECT * FROM pg_type')
...     print(types)
>>> asyncio.get_event_loop().run_until_complete(run())
```

approzium.asyncpg.pool.**create_pool**(*dsn=None*, ***, *min_size=10*, *max_size=10*, *max_queries=50000*, *max_inactive_connection_lifetime=300.0*, *setup=None*, *init=None*, *loop=None*, *authenticator=None*, ***connect_kwargs*)

Create an Asyncpg connection pool through Approzium authentication. Takes same arguments as asyncpg.create_pool in addition to the *authenticator* argument

Returns An instance of _ApproziumPool.

Example:

```
>>> import approzium
>>> from approzium.asyncpg import create_pool
>>> auth = approzium.AuthClient("myauthenticator.com:6001", disable_tls=True)
>>> pool = await create_pool(user='postgres', authenticator=auth)
>>> con = await pool.acquire()
>>> try:
...     await con.fetch('SELECT 1')
... finally:
...     await pool.release(con)
```

1.8 approzium.mysql.connector

approzium.mysql.connector.**connect**(**args*, *authenticator=None*, ***kwargs*)

Creates a MySQL connector connection through Approzium authentication. Takes the same arguments as mysql.connector.connect, in addition to the authenticator argument.

Parameters **authenticator** (approzium.AuthClient, *optional*) – AuthClient instance to be used for authentication. If not provided, the default AuthClient, if set, is used.

Raises TypeError, if no AuthClient is given and no default one is set.

Return type mysql.connector.MySQLConnection

Example:

```
>>> import approzium
>>> from approzium.mysql.connector import connect
>>> auth = approzium.AuthClient("myauthenticator.com:6001", disable_tls=True)
>>> con = connect(user="bob", host="host.com", authenticator=auth,          ...
↪ use_pure=True)
>>> # use the connection just like any other MySQL connector connection
```

Warning: Currently, only the pure Python MySQL connector implementation is supported. Therefore, you have to pass in `use_pure=True`, otherwise, an exception is raised.

class approzium.mysql.connector.pooling.**MySQLConnectionPool**(*pool_size=5*, *pool_name=None*, *pool_reset_session=True*, ***kwargs*)

Bases: mysql.connector.pooling.MySQLConnectionPool

add_connection(*cnx=None*)

Add a connection to the pool

This method instantiates a MySQLConnection using the configuration passed when initializing the MySQLConnectionPool instance or using the set_config() method. If cnx is a MySQLConnection instance, it will be added to the queue.

Raises PoolError when no configuration is set, when no more connection can be added (maximum reached) or when the connection can not be instantiated.

set_config(***kwargs*)

Set the connection configuration for MySQLConnection instances

This method sets the configuration used for creating MySQLConnection instances. See MySQLConnection for valid connection arguments.

Raises PoolError when a connection argument is not valid, missing or not supported by MySQLConnection.

1.9 approzium.pymysql

approzium.pymysql.**connect**(**args*, ***kwargs*)

Creates a PyMySQL connection through Approzium authentication. Takes the same arguments as `pymysql.connect`, in addition to the authenticator argument.

Parameters **authenticator** (*approzium.AuthClient, optional*) – AuthClient instance to be used for authentication. If not provided, the default AuthClient, if set, is used.

Raises TypeError, if no AuthClient is given and no default one is set.

Return type pymysql.connections.Connection

Example:

```
>>> import approzium
>>> from approzium.pymysql import connect
>>> auth = approzium.AuthClient("authenticatorhost:6001", disable_tls=True)
>>> con = connect("host=DB.com dbname=mydb", authenticator=auth)
>>> # use the connection just like any other PyMySQL connection
```

1.10 Opentelemetry Integration

Approzium Python SDK supports Opentelemetry integration such that all traces generated by Approzium database connections are automatically populated with attribution tags. This allows high visibility and observability across your connection.

If any example is broken, or if you'd like to add an example to this page, feel free to raise an issue on our Github repository.

1.11 Postgres Examples

Example of creating a Psycopg2 single connection and a connection pool

`psycopg2_connect.py`:

```
import approzium
from approzium import AuthClient
from approzium.psycopg2 import connect
from approzium.psycopg2.pool import ThreadedConnectionPool

auth = AuthClient(
    "authenticator:6001",
    # This is insecure, see https://approzium.org/configuration for proper use.
    disable_tls=True,
)
dsn = "host=dbmd5 dbname=db user=bob"
conn = connect(dsn, authenticator=auth)
print("Connection Established")

approzium.default_auth_client = auth
conns = ThreadedConnectionPool(1, 5, dsn)
conn = conns.getconn()
print("Connection Pool Established")
```

Example of creating a Asyncpg single connection and a connection pool

`asyncpg_connect.py`:

```
import asyncio

from approzium import AuthClient
from approzium.asyncpg import connect
from approzium.asyncpg.pool import create_pool

auth = AuthClient(
    "authenticator:6001",
    # This is insecure, see https://approzium.org/configuration for proper use.
    disable_tls=True,
)


async def run():
    conn = await connect(user="bob", database="db", host="dbmd5", authenticator=auth)
    print("Connection Established!")
    await conn.fetch("""SELECT 1""")
    await conn.close()
```

(continues on next page)

(continued from previous page)

```
    pool = await create_pool(
        user="bob", database="db", host="dbmd5", authenticator=auth
    )
    print("Connection Established!")
    async with pool.acquire() as conn:
        await conn.fetch("""SELECT 1""")


loop = asyncio.get_event_loop()
loop.run_until_complete(run())
```

1.12 MySQL Examples

Example of creating a MySQL Connector single connection and a connection pool

`mysql_connector_connect.py`:

```
from approzium import AuthClient
from approzium.mysql.connector import connect
from approzium.mysql.connector.pooling import MySQLConnectionPool

auth = AuthClient(
    "authenticator:6001",
    # This is insecure, see https://approzium.org/configuration for proper use.
    disable_tls=True,
)
conn = connect(user="bob", authenticator=auth, host="dbmysql", use_pure=True)
print("Connection Established")

cur = conn.cursor()
cur.execute("SELECT 1")
result = next(cur)
print(result)

cnxpool = MySQLConnectionPool(
    pool_name="mypool",
    pool_size=3,
    user="bob",
    host="dbmysqlsha1",
    authenticator=auth,
    use_pure=True,
)
print("Connection Pool Established")
conn = cnxpool.get_connection()
cur = conn.cursor()
cur.execute("SELECT 1")
print(result)
```

Example of creating a PyMySQL single connection:

`pymysql_connect.py`:

```
from approzium import AuthClient
from approzium.pymysql import connect
```

(continues on next page)

(continued from previous page)

```
auth = AuthClient(
    "authenticator:6001",
    # This is insecure, see https://approzium.org/configuration for proper use.
    disable_tls=True,
)
conn = connect(host="dbmysqlsha1", user="bob", db="db", authenticator=auth)
with conn.cursor() as cursor:
    cursor.execute("SELECT 1")
    result = cursor.fetchone()
    print(result)
conn.close()
```

1.13 Opentelemetry Integration Examples

psycopg2_opentelemetry.py:

```
"""This is an example that shows Approzium Opentelemetry integration. It also
integrates with a Jaeger service to export and view generated traces.
"""
from opentelemetry import trace
from opentelemetry.exporter.jaeger.thrift import JaegerExporter
from opentelemetry.instrumentation.psycopg2 import Psycopg2Instrumentor
from opentelemetry.sdk.resources import SERVICE_NAME, Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor

import approzium
import approzium.opentelemetry
from approzium.psycopg2 import connect

auth = approzium.AuthClient("authenticator:6001")
approzium.default_auth_client = auth

trace.set_tracer_provider(
    TracerProvider(resource=Resource.create({SERVICE_NAME: "approzium_service"}))
)
tracer = trace.get_tracer(__name__)

jaeger_exporter = JaegerExporter(agent_host_name="jaeger", agent_port=6831)

trace.get_tracer_provider().add_span_processor(BatchSpanProcessor(jaeger_exporter))

approzium.opentelemetry.instrument()
Psycopg2Instrumentor().instrument()

cnx = connect("host=dbmd5 dbname=db user=bob")
cursor = cnx.cursor()
with tracer.start_as_current_span("foo"):
    with tracer.start_as_current_span("bar"):
        print("Hello world!")
        cursor.execute("SELECT 1")
cursor.close()
cnx.close()
```

If you are not using Opentelemetry, you can obtain the same attribution info manually:

psycopg2_attribution_info.py:

```
"""This example shows usage of the AuthClient.attribution_info feature. It uses
a Psycopg2 connection but the same functionality is available in any supported
database driver.
"""
import approzium
from approzium.psycopg2 import connect

auth = approzium.AuthClient("authenticator:6001")
print(auth.attribution_info)
# {'authenticator_address': 'authenticator:6001',
#  'iam_arn': 'arn:aws:iam::*******:user/****',
# 'authenticated': False,
# 'num_connections': 0
# ... additional info if available (ex: EC2 instance metadata)
# }
approzium.default_auth_client = auth
dsn = "host=dbmd5 dbname=db user=bob"
conn = connect(dsn)
print(auth.attribution_info)
# {'authenticator_address': 'authenticator:6001',
#  'iam_arn': 'arn:aws:iam::*******:user/****',
# 'authenticated': True,
# 'num_connections': 1
# ...
# }
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index