```
import asyncio

import aiohttp
from aiohttp.errors import FingerprintMismatch, ClientError

from elasticsearch.exceptions import ConnectionError, ConnectionTimeout, SSLErro
r
from elasticsearch.connection import Connection
from elasticsearch.compat import urlencode

from .helpers import ensure_future

class AIOHttpConnection(Connection):
    def __init__(self, host='localhost', port=9200, http_auth=None,
            use_ssl=False, verify_certs=False, ca_certs=None, client_cert=None,
            client_key=None, loop=None, **kwargs):
        super().__init__(host=host, port=port, **kwargs)

        self.loop = asyncio.get_event_loop() if loop is None else loop

        if http_auth is not None:
            if isinstance(http_auth, str):
                http_auth = tuple(http_auth.split(':', 1))

            if isinstance(http_auth, (tuple, list)):
                http_auth = aiohttp.BasicAuth(*http_auth)

        self.session = aiohttp.ClientSession(
            auth=http_auth,
            connector=aiohttp.TCPConnector(
                loop=self.loop,
                verify_ssl=verify_certs,
                conn_timeout=self.timeout,

            )
        )

        self.base_url = 'http%s://%s:%d%s' % (
            's' if use_ssl else '',
            host, port, self.url_prefix
        )

    def close(self):
        return ensure_future(self.session.close())

    @asyncio.coroutine
    def perform_request(self, method, url, params=None, body=None, timeout=None,
 ignore=()):
        url_path = url
        if params:
            url_path = '%s?%s' % (url, urlencode(params or {}))
        url = self.base_url + url_path

        start = self.loop.time()
        response = None
        try:
            with aiohttp.Timeout(timeout or self.timeout):
                response = yield from self.session.request(method, url, data=bod
y)
                raw_data = yield from response.text()
            duration = self.loop.time() - start

        except asyncio.TimeoutError as e:
            self.log_request_fail(method, url, body, self.loop.time() - start, e
xception=e)
            raise ConnectionTimeout('TIMEOUT', str(e), e)

        except FingerprintMismatch as e:
            self.log_request_fail(method, url, body, self.loop.time() - start, e
xception=e)
            raise SSLError('N/A', str(e), e)

        except ClientError as e:
            self.log_request_fail(method, url, body, self.loop.time() - start, e
xception=e)
            raise ConnectionError('N/A', str(e), e)

        finally:
            if response is not None:
                yield from response.release()

        # raise errors based on http status codes, let the client handle those i
f needed
        if not (200 <= response.status < 300) and response.status not in ignore:
            self.log_request_fail(method, url, body, duration, response.status,
raw_data)
            self._raise_error(response.status, raw_data)

        self.log_request_success(method, url, url_path, body, response.status, r
aw_data, duration)

        return response.status, response.headers, raw_data
```