Backend.AI Client SDK for Python Documentation

출시 버전 19.03.2

Lablup Inc.

2019년 10월 11일

목차

이 문서는 Backend.AI API를 구현하는 Python용 클라이언트 소프트웨어 개발킷(SDK)에 관한 매뉴얼입니다.

CHAPTER 1

요구사항

Python 3.5.3 또는 그보다 최신 버전이 필요합니다.

python.org에서 공식 설치 패키지를 다운로드하거나, 사용 중인 운영환경에 적합한 별도의 패키지 관리자(예: homebrew, miniconda, pyenv 등)를 이용할 수 있습니다. 이 클라이언트 SDK는 Linux, macOS, Windows 환경에서 테스트되었습니다.

CHAPTER 2

시작하기

SDK 라이브러리와 도구를 다른 소프트웨어와의 충돌 없이 설치하기 위해서 별도의 Python 가상환경(virtual environment)을 만드실 것을 권장합니다.

```
$ python3 -m venv venv-backend-ai
$ source venv-backend-ai/bin/activate
(venv-backend-ai) $
```

PyPI로부터 클라이언트 라이브러리를 설치합니다.

```
(venv-backend-ai) $ pip install -U pip setuptools
(venv-backend-ai) $ pip install backend.ai-client
```

자신의 API keypair를 다음과 같이 환경변수에 설정합니다:

```
(venv-backend-ai) $ export BACKEND_ACCESS_KEY=AKIA...
(venv-backend-ai) $ export BACKEND_SECRET_KEY=...
```

그 다음엔 첫 명령어를 실행해봅니다:

```
(venv-backend-ai) $ backend.ai --help
...
(venv-backend-ai) $ backend.ai ps
...
```

보다 자세한 내용은 *클라이언트 설정*, *명령줄 예제* 및 *코드 예제*를 참고하기 바랍니다.

2.1 시작하기

2.1.1 설치하기

Linux/macOS

We recommend using pyenv to manage your Python versions and virtual environments to avoid conflicts with other Python applications.

Create a new virtual environment (Python 3.5.3 or higher) and activate it on your shell. Then run the following commands:

```
pip install -U pip setuptools
pip install -U backend.ai-client-py
```

Create a shell script `my-backendai-env.sh` like:

```
export BACKEND_ACCESS_KEY=...
export BACKEND_SECRET_KEY=...
export BACKEND_ENDPOINT=https://my-precious-cluster
```

Run this shell script before using `backend.ai` command.

Windows

We recommend using the Anaconda Navigator to manage your Python environments with a slick GUI app.

Create a new environment (Python 3.5.3 or higher) and launch a terminal (command prompt). Then run the following commands:

```
python -m pip install -U pip setuptools
python -m pip install -U backend.ai-client-py
```

Create a batch file `my-backendai-env.bat` like:

```
chcp 65001
set PYTHONIOENCODING=UTF-8
set BACKEND_ACCESS_KEY=...
set BACKEND_SECRET_KEY=...
set BACKEND_ENDPOINT=https://my-precious-cluster
```

Run this batch file before using `backend.ai` command.

Note that this batch file switches your command prompt to use the UTF-8 codepage for correct display of special characters in the console logs.

Verification

Run `backend.ai ps` command and check if it says "there is no compute sessions running" or something similar.

If you encounter error messages about "ACCESS_KEY", then check if your batch/shell scripts have the correct environment variable names.

If you encounter network connection error messages, check if the endpoint server is configured correctly and accessible.

2.1.2 코드 예제

동기 모드 실행

쿼리 모드

This is the minimal code to execute a code snippet with this client SDK.

```
import sys
from ai.backend.client import Session

with Session() as session:
    kern = session.Kernel.get_or_create('python:3.6-ubuntu18.04')
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    code = 'print("hello world")'
    mode = 'query'
    run_id = None
    while True:
        result = kern.execute(run_id, code, mode=mode)
        run_id = result['runId']  # keeps track of this particular run loop
        for rec in result.get('console', []):
            if rec[0] == 'stdout':
                print(rec[1], end='', file=sys.stdout)
            elif rec[0] == 'stderr':
                print(rec[1], end='', file=sys.stderr)
            else:
                handle_media(rec)
        sys.stdout.flush()
        if result['status'] == 'finished':
            break
        else:
            mode = 'continued'
            code = ''
    kern.destroy()
```

You need to take care of `client_token` because it determines whether to reuse kernel sessions or not. Backend.AI cloud has a timeout so that it terminates long-idle kernel sessions, but within the timeout, any kernel creation requests with the same `client_token` let Backend.AI cloud to reuse the kernel.

배치 모드

You first need to upload the files after creating the session and construct a `opts` struct.

```
import sys
from ai.backend.client import Session

with Session() as session:
    kern = session.Kernel.get_or_create('python:3.6-ubuntu18.04')
    kern.upload(['mycode.py', 'setup.py'])
    code = ''
    mode = 'batch'
    run_id = None
    opts = {
        'build': '*',  # calls "python setup.py install"
        'exec': 'python mycode.py arg1 arg2',
    }
    while True:
        result = kern.execute(run_id, code, mode=mode, opts=opts)
        opts.clear()
        run_id = result['runId']
        for rec in result.get('console', []):
            if rec[0] == 'stdout':
                print(rec[1], end='', file=sys.stdout)
            elif rec[0] == 'stderr':
                print(rec[1], end='', file=sys.stderr)
            else:
                handle_media(rec)
        sys.stdout.flush()
        if result['status'] == 'finished':
            break
        else:
            mode = 'continued'
            code = ''
    kern.destroy()
```

사용자 입력 다루기

Inside the while-loop for `kern.execute()` above, change the if-block for `result['status']` as follows:

```
...
if result['status'] == 'finished':
    break
elif result['status'] == 'waiting-input':
    mode = 'input'
    if result['options'].get('is_password', False):
        code = getpass.getpass()
    else:
        code = input()
else:
    mode = 'continued'
    code = ''
...
```

A common gotcha is to miss setting `mode = 'input'`. Be careful!

멀티미디어 출력 다루기

The `handle_media()` function used above examples would look like:

```
def handle_media(record):
    media_type = record[0]  # MIME-Type string
    media_data = record[1]  # content
    ...
```

The exact method to process `media_data` depends on the `media_type`. Currently the following behaviors are well-defined:

- For (binary-format) images, the content is a dataURI-encoded string.
- For SVG (scalable vector graphics) images, the content is an XML string.
- For `application/x-sorna-drawing`, the content is a JSON string that represents a set of vector drawing commands to be replayed the client-side (e.g., Javascript on browsers)

Asynchronous-mode Execution

The async version has all sync-version interfaces as coroutines but comes with additional features such as `stream_execute()` which streams the execution results via websockets and `stream_pty()` for interactive terminal streaming.

```
import asyncio
import json
import sys
import aiohttp
from ai.backend.client import AsyncSession

async def main():
    async with AsyncSession() as session:
        kern = await session.Kernel.get_or_create('python:3.6-ubuntu18.04',
                                                  client_token='mysession')
        code = 'print("hello world")'
        mode = 'query'
        async with kern.stream_execute(code, mode=mode) as stream:
            # no need for explicit run_id since WebSocket connection represents it!
            async for result in stream:
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
                if result.type != aiohttp.WSMsgType.TEXT:
                    continue
                result = json.loads(result.data)
                for rec in result.get('console', []):
                    if rec[0] == 'stdout':
                        print(rec[1], end='', file=sys.stdout)
                    elif rec[0] == 'stderr':
                        print(rec[1], end='', file=sys.stderr)
                    else:
                        handle_media(rec)
                sys.stdout.flush()
                if result['status'] == 'finished':
                    break
                elif result['status'] == 'waiting-input':
                    mode = 'input'
                    if result['options'].get('is_password', False):
                        code = getpass.getpass()
                    else:
                        code = input()
                    await stream.send_text(code)
                else:
                    mode = 'continued'
                    code = ''
        await kern.destroy()

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(main())
finally:
    loop.close()
```

버전 1.5에 추가.

2.1.3 클라이언트 세션

This module is the first place to begin with your Python programs that use Backend.AI API functions.

The high-level API functions cannot be used alone – you must initiate a client session first because each session provides *proxy attributes* that represent API functions and run on the session itself.

To achieve this, during initialization session objects internally construct new types by combining the *BaseFunction* class with the attributes in each API function classes, and makes the new types bound to itself. Creating new types every time when creating a new session instance may look weird, but it is the most convenient way to provide *class-methods* in the API function classes to work with specific *session instances*.

When designing your application, please note that session objects are intended to live long following the process' lifecycle, instead of to be created and disposed whenever making API requests.

class ai.backend.client.session.**BaseSession**(*, *config=None*)
The base abstract class for sessions.

class ai.backend.client.session.**Session**(*, *config=None*)
An API client session that makes API requests synchronously. You may call (almost) all function proxy methods like a plain Python function. It provides a context manager interface to ensure closing of the session upon errors and scope exits.

Admin
The *Admin* function proxy bound to this session.

Agent
The *Agent* function proxy bound to this session.

Image
The `Image` function proxy bound to this session.

Kernel
The *Kernel* function proxy bound to this session.

KeyPair
The *KeyPair* function proxy bound to this session.

Manager
The *Manager* function proxy bound to this session.

Resource
The `Resource` function proxy bound to this session.

ResourcePolicy
The `ResourcePolicy` function proxy bound to this session.

VFolder
The *VFolder* function proxy bound to this session.

close()
Terminates the session. It schedules the `close()` coroutine of the underlying aiohttp session and then enqueues a sentinel object to indicate termination. Then it waits until the worker thread to self-terminate by joining.

class ai.backend.client.session.**AsyncSession**(*, *config=None*)
An API client session that makes API requests asynchronously using coroutines. You may call all function proxy methods like a coroutine. It provides an async context manager interface to ensure closing of the session upon errors and scope exits.

Admin
The *Admin* function proxy bound to this session.

Agent
The *Agent* function proxy bound to this session.

Image
The `Image` function proxy bound to this session.

Kernel
The *Kernel* function proxy bound to this session.

KeyPair
The *KeyPair* function proxy bound to this session.

Manager
The *Manager* function proxy bound to this session.

Resource
The `Resource` function proxy bound to this session.

ResourcePolicy
The `ResourcePolicy` function proxy bound to this session.

VFolder
The *VFolder* function proxy bound to this session.

2.1.4 클라이언트 설정

Backend.AI API 설정에는 endpoint URL과 API keypair (access 및 secret key) 등이 포함됩니다.

설정 방법은 2가지가 있습니다:

1. 이 SDK를 사용하는 여러분의 프로그램이 실행되기 전 환경변수를 미리 설정해두기
2. 직접 *APIConfig* 인스턴스를 만들고 그로부터 세션을 초기화하기

다음과 같은 환경변수들을 사용할 수 있습니다:

- BACKEND_ENDPOINT
- BACKEND_ACCESS_KEY
- BACKEND_SECRET_KEY
- BACKEND_VFOLDER_MOUNTS

다른 설정들은 기본값을 사용합니다.

클라이언트측 Jupyter 통합 플러그인을 사용하는 경우, BACKEND_VFOLDER_MOUNTS 변수는 가상폴더를 노트북 커널 안에서 사용하기 위한 유일한 방법입니다.

ai.backend.client.config.**get_env**(*key*, *default=None*, *clean=<function <lambda>>*)
환경변수에서 설정값을 가져옵니다. 설정 이름을 대문자화한 후 먼저 "BACKEND_" 접두어를 붙여서 찾아보고, 없으면 "SORNA_"를 붙여서 찾아봅니다.

매개변수

- **key** (str) – 설정 이름
- **default** (Optional[Any]) – 환경변수에 대응하는 값이 없을 때 반환되는 기본값
- **clean** (Callable[[str], Any]) – A single-argument function that is applied to the result of lookup (in both successes and the default value for failures). The default is returning the value as-is.

반환값 *clean* 함수를 거쳐 처리된 값

ai.backend.client.config.**get_config**()
Returns the configuration for the current process. If there is no explicitly set *APIConfig* instance, it will generate a new one from the current environment variables and defaults.

ai.backend.client.config.**set_config**(*conf*)
현재 프로세스 전체에서 사용될 전역 설정을 지정합니다.

class ai.backend.client.config.**APIConfig**(*, *endpoint=None*, *version=None*, *user_agent=None*, *access_key=None*, *secret_key=None*, *hash_type=None*, *vfolder_mounts=None*, *skip_sslcert_validation=None*)
Represents a set of API client configurations. The access key and secret key are mandatory – they must be set in either environment variables or as the explicit arguments.

매개변수

- **endpoint** (Union[URL, str, None]) – API 서버의 주소
- **version** (Optional[str]) – API 프로토콜 버전
- **user_agent** (Optional[str]) – User-Agent HTTP 헤더로 전송될 사용자 에이전트 문자열
- **access_key** (Optional[str]) – API access key
- **secret_key** (Optional[str]) – API secret key
- **hash_type** (Optional[str]) – API 요청별 인증 서명 생성 알고리즘을 위한 해쉬 알고리즘 종류
- **vfolder_mounts** (Optional[Iterable[str]]) – Kernel.get_or_create()와 같이 세션 생성 API 호출 시 자동으로 탑재될 가상폴더의 목록 (반드시 요청자의 access key가 소유 또는 접근 가능한 것이어야 함)

DEFAULTS = {'endpoint': 'https://api.backend.ai', 'hash_type': 'sha256', 'version':
access key와 secret key를 제외한 기본값

2.2 명령줄 인터페이스

2.2.1 예제

참고: 자세한 명령어 사용법들은 `-h`, `--help` 인자를 사용하여 각 명령어의 세부 도움말을 참고하십시오.

현재 실행 중인 세션 목록 보기

```
backend.ai ps
```

이 명령은 사실 다음 명령어의 짧은 별칭입니다:

```
backend.ai admin sessions
```

간단한 코드 실행하기

다음 명령어는 Python 세션을 띄우고 그 안에서 `-c` 옵션으로 넘겨진 코드를 바로 실행합니다. `--rm` 옵션은 실행이 완료되자마자 클라이언트가 자동으로 세션을 종료시키도록 지시합니다.

```
backend.ai run --rm -c 'print("hello world")' python
```

다음 명령어는 Python 세션을 띄우고 `./myscript.py` 파일로 전달된 코드를 그 안에서 실행합니다. 이때 `--exec` 옵션으로 지정한 shell 명령어를 이용하여 실행합니다.

```
backend.ai run --rm --exec 'python myscript.py arg1 arg2' \
           python ./myscript.py
```

가속기를 활용한 세션 사용하기

다음 명령어는 Python TensorFlow 세션을 가상 GPU 0.5개를 활용하도록 지정하여 띄우고, 그 안에서 `./mygpucode.py` 파일을 실행합니다.

```
backend.ai run --rm -r gpu=0.5 \
           python-tensorflow ./mygpucode.py
```

실행 중인 세션 종료하기

`--rm` 옵션 없이 `run` 명령을 실행하거나 `start` 명령을 사용한 경우 여러분의 연산 세션은 미리 설정된 시간 (기본 30분) 동안 계속 떠있습니다.그렇게 대기 중인 세션들은 `backend.ai ps` 명령으로 조회할 수 있습니다. 다음 명령어를 활용하여 그러한 세션들을 세션 ID를 이용해 종료합니다. 세션 ID를 여러 개 적어서 여러 세션을 한꺼번에 종료할 수도 있습니다.

```
backend.ai rm <sessionID>
```

Jupyter Notebook용 세션을 만들고 접속하기

다음 명령어는 "mysession"이라 이름붙여진 Python 세션을 띄웁니다. 이때 띄우기만 하고 코드를 바로 실행하지는 않습니다. 이어서, 그 안에서 실행되는 "jupyter" 서비스에 접속하는 로컬 프록시를 TCP 포트 9900번에 실행합니다. `start` 명령은 생성된 연산 세션이 제공하는 응용 서비스들을 보여주므로, 여러분은 이어서 `app`을 실행할 때 어떤 것으로 접속할지 선택할 수 있습니다.

```
backend.ai start -t mysession python
backend.ai app -p 9900 mysession jupyter
```

app 명령은 한번 실행되면 사용자가 적절한 소프트웨어로 접속하기를 기다립니다. Jupyter 서비스의 경우, 여러분이 선호하는 웹브라우저를 이용하여 원래 Jupyter Notebook 환경을 사용하듯이 해당 주소로 접속하면 됩니다. app 명령을 중지하려면 Ctrl+C 단축키를 누르거나 SIGINT 시그널을 보냅니다.

가상폴더를 사용한 연산 세션 실행하기

다음 명령어는 "mydata1"이라는 이름의 가상 폴더를 생성하고, 그 안에 ./bigdata.csv라는 이름의 파일을 업로드합니다.

```
backend.ai vfolder create mydata1
backend.ai vfolder upload mydata1 ./bigdata.csv
```

다음 명령어는 "mydata1"이란 가상 폴더가 탑재된 Python 세션을 하나 띄웁니다. 코드 실행 옵션은 편의상 생략되었습니다. 이어서 두번째 명령은 여러분의 코드가 생성한 ./bigresult.txt 파일을 "mydata1" 가상 폴더로부터 다운로드합니다.

```
backend.ai run --rm -m mydata1 python ...
backend.ai vfolder download mydata1 ./bigresult.txt
```

세션 안에서 실행되는 여러분의 코드에서는 가상폴더를 /home/work/mydata1 이라는 일반 디렉토리처럼 그대로 사용할 수 있습니다. (이때 기본 활성 작업 디렉토리는 /home/work입니다.)

병렬 실험을 위한 연산 세션 실행하기

(준비 중입니다)

2.3 고수준 함수 인터페이스

2.3.1 관리자용 함수

class ai.backend.client.admin.**Admin**

관리자 GraphQL 쿼리를 날리고 받아오는 함수형 인터페이스를 제공합니다.

> **참고:** Depending on the privilege of your API access key, you may or may not have access to querying/mutating server-side resources of other users.

session = None

이 함수 클래스가 사용할 클라이언트 세션 인스턴스

2.3.2 에이전트 함수

class ai.backend.client.agent.**Agent**

서버측 에이전트들에 관한 정보를 조회하기 위한 Admin.query() 함수의 단축 버전입니다.

> **참고:** 이 함수 클래스의 모든 메소드들은 사용 중인 API access key가 관리자 권한을 가지고 있어야 작동합니다.

session = None

이 함수 클래스가 사용할 클라이언트 세션 인스턴스

2.3.3 커널 함수

class ai.backend.client.kernel.**Kernel**(*kernel_id*, *owner_access_key=None*)
연산 세션들을 관리하면서 다양한 상호작용을 제공합니다.

Backend.AI의 개발이 진행됨에 따라 '커널'이라는 용어는 '연산 세션'이라는 용어로 대체되었지만, 클라이언트 세션과의 혼동을 피하고 과거 코드와의 호환성을 위해 여전히 이 함수 클래스는 커널이라는 이름을 사용하고 있습니다.

여러 개의 다중 컨테이너로 이뤄진 연산 세션에서는, `destroy()` 및 `restart()` 메소드를 제외한 모든 메소드는 세션의 마스터 컨테이너에게만 유효합니다. 따라서 여러 컨테이너가 동일한 데이터를 바라보거나 분산 처리하도록 하기 위해서는 가상폴더 탑재 옵션을 활용하여야 합니다. 현재 이러한 작업은 사용자의 몫입니다. (추후 업데이트로 편의성 개선 예정)

session = None
이 함수 클래스가 사용할 클라이언트 세션 인스턴스

stream_pty()
Opens a pseudo-terminal of the kernel (if supported) streamed via websockets.

반환 형식 *StreamPty*

반환값 a *StreamPty* object.

stream_execute(*code=''*, ***, *mode='query'*, *opts=None*)
Executes a code snippet in the streaming mode. Since the returned websocket represents a run loop, there is no need to specify *run_id* explicitly.

반환 형식 *WebSocketResponse*

class ai.backend.client.kernel.**StreamPty**(*session*, *underlying_ws*)
A derivative class of *WebSocketResponse* which provides additional functions to control the terminal.

2.3.4 KeyPair 함수

class ai.backend.client.keypair.**KeyPair**(*access_key*)
Keypair들을 다룹니다.

session = None
이 함수 클래스가 사용할 클라이언트 세션 인스턴스

2.3.5 매니저 함수

class ai.backend.client.manager.**Manager**
게이트웨이 및 매니저 서버의 상태를 관리합니다.

버전 18.12에 추가.

session = None
이 함수 클래스가 사용할 클라이언트 세션 인스턴스

2.3.6 가상폴더 함수

class ai.backend.client.vfolder.**VFolder**(*name*)

session = None
이 함수 클래스가 사용할 클라이언트 세션 인스턴스

2.4 저수준 레퍼런스

2.4.1 기반 함수

This module defines a few utilities that ease complexities to support both synchronous and asynchronous API functions, using some tricks with Python metaclasses.

Unless your are contributing to the client SDK, probably you won't have to use this module directly.

class ai.backend.client.base.**APIFunctionMeta**(*name*, *bases*, *attrs*, ***kwargs*)
Converts all methods marked with *api_function()* into session-aware methods that are either plain Python functions or coroutines.

mro() → list
return a type's method resolution order

class ai.backend.client.base.**BaseFunction**
The class used to build API functions proxies bound to specific session instances.

@ai.backend.client.base.**api_function**(*meth*)
Mark the wrapped method as the API function method.

2.4.2 요청 API

This module provides low-level API request/response interfaces based on aiohttp.

Depending on the session object where the request is made from, *Request* and *Response* differentiate their behavior: works as plain Python functions or returns awaitables.

class ai.backend.client.request.**Request**(*session*, *method='GET'*, *path=None*, *content=None*, ***, *content_type=None*, *params=None*, *reporthook=None*)
The API request object.

with async with fetch(***kwargs*) **as Response**
Sends the request to the server and reads the response.

You may use this method either with plain synchronous Session or AsyncSession. Both the followings patterns are valid:

```
from ai.backend.client.request import Request
from ai.backend.client.session import Session

with Session() as sess:
  rqst = Request(sess, 'GET', ...)
  with rqst.fetch() as resp:
    print(resp.text())
```

```
from ai.backend.client.request import Request
from ai.backend.client.session import AsyncSession

async with AsyncSession() as sess:
  rqst = Request(sess, 'GET', ...)
  async with rqst.fetch() as resp:
    print(await resp.text())
```

반환 형식 *FetchContextManager*

async with connect_websocket(***kwargs*) **as WebSocketResponse or its derivatives**
Creates a WebSocket connection.

> **경고:** This method only works with *AsyncSession*.

반환 형식 *WebSocketContextManager*

set_content (*value*, *, *content_type=None*)
Sets the content of the request.

set_json (*value*)
A shortcut for set_content() with JSON objects.

attach_files (*files*)
Attach a list of files represented as AttachedFile.

class ai.backend.client.request.**Response** (*session*, *underlying_response*, *, *async_mode=False*)
Represents the Backend.AI API response. Also serves as a high-level wrapper of aiohttp.ClientResponse.

The response objects are meant to be created by the SDK, not the callers.

`text()`, `json()` methods return the resolved content directly with plain synchronous Session while they return the coroutines with AsyncSession.

class ai.backend.client.request.**WebSocketResponse** (*session*, *underlying_ws*)
A high-level wrapper of aiohttp.ClientWebSocketResponse.

class ai.backend.client.request.**FetchContextManager** (*session*, *rqst_ctx*, *, *response_cls=<class 'ai.backend.client.request.Response'>*, *check_status=True*)
The context manager returned by *Request.fetch()*.

It provides both synchronouse and asynchronous contex manager interfaces.

class ai.backend.client.request.**WebSocketContextManager** (*session*, *ws_ctx*, *, *on_enter=None*, *response_cls=<class 'ai.backend.client.request.WebSocketResponse'*
The context manager returned by *Request.connect_websocket()*.

class ai.backend.client.request.**AttachedFile** (*filename*, *stream*, *content_type*)
A struct that represents an attached file to the API request.

매개변수

- **filename** (*str*) – The name of file to store. It may include paths and the server will create parent directories if required.
- **stream** (*Any*) – A file-like object that allows stream-reading bytes.
- **content_type** (*str*) – The content type for the stream. For arbitrary binary data, use "application/octet-stream".

count (*value*) → integer – return number of occurrences of value

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

2.4.3 예외 클래스들

class ai.backend.client.exceptions.**BackendError**
Exception type to catch all ai.backend-related errors.

with_traceback ()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class ai.backend.client.exceptions.**BackendAPIError**(*status*, *reason*, *data*)
Exceptions returned by the API gateway.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class ai.backend.client.exceptions.**BackendClientError**
Exceptions from the client library, such as argument validation errors.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

CHAPTER 3

색인과 표

- genindex
- modindex
- search

Python 모듈 목록

a

색인

R

S

V

W