Backend.AI Client SDK for Python Documentation

Release 20.03.10

Lablup Inc.

Mar 04, 2021

This is the documentation for the Python Client SDK which implements the Backend.AI API.

QUICKSTART

Python 3.6 or higher is required.

You can download its official installer from python.org, or use a 3rd-party package/version manager such as homebrew, miniconda, or pyenv. It works on Linux, macOS, and Windows.

We recommend to create a virtual environment for isolated, unobtrusive installation of the client SDK library and tools.

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

Then install the client library from PyPI.

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

Set your API keypair as environment variables:

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

And then try the first commands:

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

Check out more details about *client configuration*, the command-line examples, and *SDK code examples*.

GETTING STARTED

2.1 Installation

2.1.1 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 3.6 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
export BACKEND_ENDPOINT_TYPE=api
```

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

Note: The console-server users should set `BACKEND_ENDPOINT_TYPE` to `session`. For details, check out *the client configuration document*.

2.1.2 Windows

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

Create a new environment (Python 3.6 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
set BACKEND_ENDPOINT_TYPE=api
```

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.

2.1.3 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.

2.2 Client Configuration

The configuration for Backend.AI API includes the endpoint URL prefix, API keypairs (access and secret keys), and a few others.

There are two ways to set the configuration:

1. Setting environment variables before running your program that uses this SDK. This applies to the command-line interface as well.

2. Manually creating *APIConfig* instance and creating sessions with it.

The list of configurable environment variables are:

- BACKEND_ENDPOINT

- BACKEND_ENDPOINT_TYPE

- BACKEND_ACCESS_KEY

- BACKEND_SECRET_KEY

- BACKEND_VFOLDER_MOUNTS

Please refer the parameter descriptions of *APIConfig*'s constructor for what each environment variable means and what value format should be used.

COMMAND-LINE INTERFACE

3.1 Configuration

Note: Please consult the detailed usage in the help of each command (use `-h` or `--help` argument to display the manual).

Check out *the client configuration* for configurations via environment variables.

3.1.1 Session Mode

When the endpoint type is `"session"`, you must explicitly login and logout into/from the console server.

```
$ backend.ai login
Username: myaccount@example.com
Password:
✓ Login succeeded.

$ backend.ai ...  # any commands

$ backend.ai logout
✓ Logout done.
```

3.1.2 API Mode

After setting up the environment variables, just run any command:

```
$ backend.ai ...
```

3.1.3 Checking out the current configuration

Run the following command to list your current active configurations.

```
$ backend.ai config
```

3.2 Compute Sessions

Note: Please consult the detailed usage in the help of each command (use `-h` or `--help` argument to display the manual).

3.2.1 Listing sessions

List the session owned by you with various status filters. The most recently status-changed sessions are listed first. To prevent overloading the server, the result is limited to the first 10 sessions and it provides a separate `--all` option to paginate further sessions.

```
backend.ai ps
```

The `ps` command is an alias of the following `admin sessions` command. If you have the administrator privilege, you can list sessions owned by other users by adding `--access-key` option here.

```
backend.ai admin sessions
```

Both commands offer options to set the status filter as follows. For other options, please consult the output of `--help`.

Option	Included Session Status
(no option)	PENDING, PREPARING, RUNNING, RESTARTING, TERMINATING, RESIZING, SUSPENDED, and ERROR.
`--running`	PREPARING, PULLING, and RUNNING.
`--dead`	CANCELLED and TERMINATED.

Both commands offer options to specify which fields of sessions should be printed as follows.

Option	Included Session Fields
(no option)	`Session ID`, `Owner`, `Image`, `Type`, `Status`, `Status Info`, `Last updated`, and `Result`.
`--id-only`	`Session ID`.
`--detail`	`Session ID`, `Owner`, `Image`, `Type`, `Status`, `Status Info`, `Last updated`, `Result`, `Tag`, `Created At`, `Occupied Resource`, `Used Memory (MiB)`, `Max Used Memory (MiB)`, and `CPU Using (%)`.
`-f, --format`	Specified fields by user.

Note: Fields for `-f`/`--format` option can be displayed by specifying comma-separated parameters.

Available parameters for this option are: `id`, `status`, `status_info`, `created_at`, `last_updated`, `result`, `image`, `type`, `task_id`, `tag`, `occupied_slots`, `used_memory`, `max_used_memory`, `cpu_using`.

For example:

```
backend.ai admin session --format id,status,cpu_using
```

3.2.2 Running simple sessions

The following command spawns a Python session and executes the code passed as `-c` argument immediately. `--rm` option states that the client automatically terminates the session after execution finishes.

```
backend.ai run --rm -c 'print("hello world")' python:3.6-ubuntu18.04
```

Note: By default, you need to specify language with full version tag like `python:3.6-ubuntu18.04`. Depending on the Backend.AI admin's language alias settings, this can be shortened just as `python`. If you want to know defined language aliases, contact the admin of Backend.AI server.

The following command spawns a Python session and executes the code passed as `./myscript.py` file, using the shell command specified in the `--exec` option.

```
backend.ai run --rm --exec 'python myscript.py arg1 arg2' \
        python:3.6-ubuntu18.04 ./myscript.py
```

Please note that your `run` command may hang up for a very long time due to queueing when the cluster resource is not sufficiently available.

To avoid indefinite waiting, you may add `--enqueue-only` to return immediately after posting the session creation request.

Note: When using `--enqueue-only`, the codes are *NOT* executed and relevant options are ignored. This makes the `run` command to the same of the `start` command.

Or, you may use `--max-wait` option to limit the maximum waiting time. If the session starts within the given `--max-wait` seconds, it works normally, but if not, it returns without code execution like when used `--enqueue-only`.

To watch what is happening behind the scene until the session starts, try `backend.ai events <sessionID>` to receive the lifecycle events such as its scheduling and preparation steps.

3.2.3 Running sessions with accelerators

Use one or more `-r` options to specify resource requirements when using `backend.ai run` and `backend.ai start` commands.

For instance, the following command spawns a Python TensorFlow session using a half of virtual GPU device, 4 CPU cores, and 8 GiB of the main memory to execute `./mygpucode.py` file inside it.

```
backend.ai run --rm \
          -r cpu=4 -r mem=8g -r cuda.shares=2 \
          python-tensorflow:1.12-py36 ./mygpucode.py
```

3.2.4 Terminating or cancelling sessions

Without `--rm` option, your session remains alive for a configured amount of idle timeout (default is 30 minutes). You can see such sessions using the `backend.ai ps` command. Use the following command to manually terminate them via their session IDs. You may specifcy multiple session IDs to terminate them at once.

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

If you terminate `PENDING` sessions which are not scheduled yet, they are cancelled.

3.3 Container Applications

Note: Please consult the detailed usage in the help of each command (use `-h` or `--help` argument to display the manual).

3.3.1 Starting a session and connecting to its Jupyter Notebook

The following command first spawns a Python session named "mysession" without running any code immediately, and then executes a local proxy which connects to the "jupyter" service running inside the session via the local TCP port 9900. The `start` command shows application services provided by the created compute session so that you can choose one in the subsequent `app` command. In the `start` command, you can specify detailed resource options using `-r` and storage mounts using `-m` parameter.

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

Once executed, the `app` command waits for the user to open the displayed address using appropriate application. For the jupyter service, use your favorite web browser just like the way you use Jupyter Notebooks. To stop the `app` command, press `Ctrl+C` or send the `SIGINT` signal.

3.3.2 Accessing sessions via a web terminal

All Backend.AI sessions expose an intrinsic application named `"ttyd"`. It is an web application that embeds xterm.js-based full-screen terminal that runs on web browsers.

```
backend.ai start -t mysession ...
backend.ai app -b 9900 mysession ttyd
```

Then open `http://localhost:9900` to access the shell in a fully functional web terminal using browsers. The default shell is `/bin/bash` for Ubuntu/CentOS-based images and `/bin/ash` for Alpine-based images with a fallback to `/bin/sh`.

Note: This shell access does *NOT* grant your root access. All compute session processes are executed as the user privilege.

3.3.3 Accessing sessions via native SSH/SFTP

Backend.AI offers direct access to compute sessions (containers) via SSH and SFTP, by auto-generating host identity and user keypairs for all sessions. All Baceknd.AI sessions expose an intrinsic application named "sshd" like "ttyd".

To connect your sessions with SSH, first prepare your session and download an auto-generated SSH keypair named id_container. Then start the service port proxy ("app" command) to open a local TCP port that proxies the SSH/SFTP traffic to the compute sessions:

```
$ backend.ai start -t mysess ...
$ backend.ai download mysess id_container
$ mv id_container ~/.ssh
$ backend.ai app mysess sshd -b 9922
```

In another terminal on the same PC, run your ssh client like:

```
$ ssh -o StrictHostKeyChecking=no \
>     -o UserKnownHostsFile=/dev/null \
>     -i ~/.ssh/id_container \
>     work@localhost -p 9922
Warning: Permanently added '[127.0.0.1]:9922' (RSA) to the list of known hosts.
f310e8dbce83:~$
```

This SSH port is also compatible with SFTP to browse the container's filesystem and to upload/download large-sized files.

You could add the following to your ~/.ssh/config to avoid type extra options every time.

```
Host localhost
  User work
  IdentityFile ~/.ssh/id_container
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
```

```
$ ssh localhost -p 9922
```

> **Warning:** Since the SSH keypair is auto-generated every time when your launch a new compute session, you need to download and keep it separately for each session.

To use your own SSH private key across all your sessions without downloading the auto-generated one every time, create a vfolder named .ssh and put the authorized_keys file that includes the public key. The keypair and .ssh directory permissions will be automatically updated by Backend.AI when the session launches.

```
$ ssh-keygen -t rsa -b 2048 -f id_container
$ cat id_container.pub > authorized_keys
$ backend.ai vfolder create .ssh
$ backend.ai vfolder upload .ssh authorized_keys
```

3.4 Storage Management

Note: Please consult the detailed usage in the help of each command (use `-h` or `--help` argument to display the manual).

Backend.AI abstracts shared network storages into per-user slices called **"virtual folders"** (aka **"vfolders"**), which can be shared between users and user group members.

3.4.1 Creating vfolders and managing them

The command-line interface provides a set of subcommands under `backend.ai vfolder` to manage vfolders and files inside them.

To list accessible vfolders including your own ones and those shared by other users:

```
$ backend.ai vfolder list
```

To create a virtual folder named "mydata1":

```
$ backend.ai vfolder create mydata1 mynas
```

The second argument `mynas` corresponds to the name of a storage host. To list up storage hosts that you are allowed to use:

```
$ backend.ai vfolder list-hosts
```

To delete the vfolder completey:

```
$ backend.ai vfolder delete mydata1
```

3.4.2 File transfers and management

To upload a file from the current working directory into the vfolder:

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

To download a file from the vfolder into the current working directory:

```
$ backend.ai vfolder download mydata1 ./bigresult.txt
```

To list files in the vfolder's specific path:

```
$ backend.ai vfolder ls mydata1 .
```

To delete files in the vfolder:

```
$ backend.ai vfolder rm mydata1 ./bigdata.csv
```

> **Warning:** All file uploads and downloads overwrite existing files and all file operations
> are irreversible.

3.4.3 Running sessions with storages

The following command spawns a Python session where the virtual folder "mydata1"
is mounted. The execution options are omitted in this example. Then, it downloads `./
bigresult.txt` file (generated by your code) from the "mydata1" virtual folder.

```
$ backend.ai vfolder upload mydata1 ./bigdata.csv
$ backend.ai run --rm -m mydata1 python:3.6-ubuntu18.04 ...
$ backend.ai vfolder download mydata1 ./bigresult.txt
```

In your code, you may access the virtual folder via `/home/work/mydata1` (where the default
current working directory is `/home/work`) just like a normal directory.

By reusing the same vfolder in subsequent sessions, you do not have to donwload the result
and upload it as the input for next sessions, just keeping them in the storage.

3.4.4 Creating default files for kernels

Backend.AI has a feature called 'dotfile', created to all the kernels user spawns. As you can guess, dotfile's path should start with .. The following command creates dotfile named .aws/config with permission *755*. This file will be created under /home/work every time user spawns Backend.AI kernel.

```
$ backend.ai dotfile create .aws/config < ~/.aws/config
```

3.5 Advanced Code Execution

Note: Please consult the detailed usage in the help of each command (use -h or --help argument to display the manual).

3.5.1 Running concurrent experiment sessions

In addition to single-shot code execution as described in *Running simple sessions*, the run command offers concurrent execution of multiple sessions with different parameters interpolated in the execution command specified in --exec option and environment variables specified as -e / --env options.

To define variables interpolated in the --exec option, use --exec-range. To define variables interpolated in the --env options, use --env-range.

Here is an example with environment variable ranges that expands into 4 concurrent sessions.

```
backend.ai run -c 'import os; print("Hello world, {}".format(os.environ["CASENO"]))
↪' \
    -r cpu=1 -r mem=256m \
    -e 'CASENO=$X' \
    --env-range=X=case:1,2,3,4 \
    lablup/python:3.6-ubuntu18.04
```

Both range options accept a special form of argument: "range expressions". The front part of range option value consists of the variable name used for interpolation and an equivalence sign (=). The rest of range expressions have the following three types:

Expression	Interpretation
`case:CASE1,` `CASE2,...,CASEN`	A list of discrete values. The values may be either string or numbers.
`linspace:START,` `STOP,POINTS`	An inclusive numerical range with discrete points, in the same way of `numpy.linspace()`. For example, `linspace:1,2,3` generates a list of three values: 1, 1.5, and 2.
`range:START,` `STOP,STEP`	A numerical range with the same semantics of Python's `range()`. For example, `range:1,6,2` generates a list of values: 1, 3, and 5.

If you specify multiple occurrences of range options in the `run` command, the client spawns sessions for *all possible combinations* of all values specified by each range.

Note: When your resource limit and cluster's resource capacity cannot run all spawned sessions at the same time, some of sessions may be queued and the command may take a long time to finish.

Warning: Until all cases finish, the client must keep its network connections to the server alive because this feature is implemented in the client-side. Server-side batch job scheduling is under development!

3.6 Session Templates

3.6.1 Creating and starting session template

Users may define commonly used set of session creation parameters as reusable templates.

A session template includes common session parameters such as resource slots, vfolder mounts, the kernel iamge to use, and etc. It also support an extra feature that automatically clones a Git repository upon startup as a bootstrap command.

The following sample shows how a session template looks like:

```
---
api_version: v1
kind: taskTemplate
metadata:
  name: template1234
  tag: example-tag
spec:
  kernel:
```

(continues on next page)

```yaml
  environ:
    MYCONFIG: XXX
  git:
    branch: '19.09'
    commit: 10daee9e328876d75e6d0fa4998d4456711730db
    repository: https://github.com/lablup/backend.ai-agent
    destinationDir: /home/work/baiagent
  image: python:3.6-ubuntu18.04
  resources:
    cpu: '2'
    mem: 4g
  mounts:
    hostpath-test: /home/work/hostDesktop
    test-vfolder:
  sessionType: interactive
```

The `backend.ai sesstpl` command set provides the basic CRUD operations of user-specific session templates.

The `create` command accepts the YAML content either piped from the standard input or read from a file using `-f` flag:

```
$ backend.ai sesstpl create < session-template.yaml
# -- or --
$ backend.ai sesstpl create -f session-template.yaml
```

Once the session template is uploaded, you may use it to start a new session:

```
$ backend.ai start-template <templateId>
```

with substituting `<templateId>` to your template ID.

Other CRUD command examples are as follows:

```
$ backend.ai sesstpl update <templateId> < session-template.yaml
$ backend.ai sesstpl list
$ backend.ai sesstpl get <templateId>
$ backend.ai sesstpl delete <templateId>
```

3.6.2 Full syntax for task template

```
---
api_version or apiVersion: str, required
kind: Enum['taskTemplate', 'task_template'], required
metadata: required
  name: str, required
  tag: str (optional)
spec:
  type or sessionType: Enum['interactive', 'batch'] (optional), default=interactive
  kernel:
    image: str, required
    environ: map[str, str] (optional)
    run: (optional)
      bootstrap: str (optional)
      stratup or startup_command or startupCommand: str (optional)
    git: (optional)
      repository: str, required
      commit: str (optional)
      branch: str (optional)
      credential: (optional)
        username: str
        password: str
      destination_dir or destinationDir: str (optional)
  mounts: map[str, str] (optional)
  resources: map[str, str] (optional)
```

DEVELOPER REFERENCE

4.1 Developer Guides

4.1.1 Client Session

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*, *proxy_mode=False*)

> The base abstract class for sessions.

> **proxy_mode**
>> If set True, it skips API version negotiation when opening the session.
>>
>>> **Return type** bool

> **abstractmethod open()**
>> Initializes the session and perform version negotiation.
>>
>>> **Return type** Union[None, Awaitable[None]]

> **abstractmethod close()**
>> Terminates the session and releases underlying resources.

> > **Return type** Union[None, Awaitable[None]]

> **closed**
> > Checks if the session is closed.

> > > **Return type** bool

> **config**
> > The configuration used by this session object.

> > > **Return type** *APIConfig*

class ai.backend.client.session.**Session**(*, *config=None, proxy_mode=False*)

> A context manager for API client sessions that makes API requests synchronously. You may call simple request-response APIs like a plain Python function, but cannot use streaming APIs based on WebSocket and Server-Sent Events.

> **closed**
> > Checks if the session is closed.

> > > **Return type** bool

> **config**
> > The configuration used by this session object.

> > > **Return type** *APIConfig*

> **proxy_mode**
> > If set True, it skips API version negotiation when opening the session.

> > > **Return type** bool

> **open**()
> > Initializes the session and perform version negotiation.

> > > **Return type** None

> **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.

> > > **Return type** None

> **worker_thread**
> > The thread that internally executes the asynchronous implementations of the given API functions.

class ai.backend.client.session.**AsyncSession**(*, config=None, proxy_mode=False*)

> A context manager for API client sessions that makes API requests asynchronously. You may call all APIs as coroutines. WebSocket-based APIs and SSE-based APIs returns special response types.

closed

 Checks if the session is closed.

  Return type bool

config

 The configuration used by this session object.

  Return type *APIConfig*

proxy_mode

 If set True, it skips API version negotiation when opening the session.

  Return type bool

open()

 Initializes the session and perform version negotiation.

  Return type Awaitable[None]

close()

 Terminates the session and releases underlying resources.

  Return type Awaitable[None]

4.1.2 Examples

Synchronous-mode execution

Query mode

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

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

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

<div align="right">(continues on next page)</div>

```python
        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.

Batch mode

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

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

with Session() as session:
    kern = session.ComputeSession.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()
```

Handling user inputs

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!

Handling multi-media outputs

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.

```python
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.ComputeSession.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:
                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 = ''
```

```python
        await kern.destroy()


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

New in version 1.5.

4.1.3 Testing

Unit Tests

Unit tests perform function-by-function tests to ensure their individual functionality. This test suite runs without depending on the server-side and thus it is executed in Travis CI for every push.

How to run

```
$ python -m pytest -m 'not integration' tests
```

Integration Tests

Integration tests combine multiple invocations of high-level interfaces to make underlying API requests to a running gateway server to test the full functionality of the client as well as the manager.

They are marked as "integration" using the `@pytest.mark.integration` decorator to each test case.

> **Warning:** The integration tests actually make changes to the target gateway server and agents. If some tests fail, those changes may remain in an inconsistent state and requires a manual recovery such as resetting the database and populating fixtures again, though the test suite tries to clean up them properly.
>
> So, DO NOT RUN it against your production server.

Prerequisite

Please refer the README of the manager and agent repositories to set up them. To avoid an indefinite waiting time for pulling Docker images:

- (manager) `python -m ai.backend.manager.cli etcd rescan-images`

- (agent) `docker pull`

    - `lablup/python:3.6-ubuntu18.04`

    - `lablup/lua:5.3-alpine3.8`

The manager must also have at least the following active suerp-admin account in the `default` domain and the `default` group.

- Example super-admin account:

    - User ID: admin@lablup.com

    - Password wJalrXUt

    - Access key: AKIAIOSFODNN7EXAMPLE

    - Secret key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

One or more `testing-XXXX` domain, one or more `testing-XXXX` groups, and one ore more dummy users are created and used during the tests and destroyed after running tests. XXXX will be filled with random identifiers.

Tip: The *halfstack* configuration and the `example-keypairs.json` fixture is compatible with this integration test suite.

How to run

Execute the gateway and at least one agent in their respective virtualenvs and hosts:

```
$ python -m ai.backend.client.gateway.server
$ python -m ai.backend.client.agent.server
$ python -m ai.backend.client.agent.watcher
```

Then run the tests:

```
$ export BACKEND_ENDPOINT=...
$ python -m pytest -m 'integration' tests
```

4.2 High-level Function Reference

4.2.1 Admin Functions

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

> Provides the function interface for making admin GrapQL queries.

> **Note:** 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.

> **classmethod await query**()
>
> > classmethod(function) -> method
> >
> > Convert a function to be a class method.
> >
> > A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
> >
> > > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> > >
> > > > . . .
> >
> > It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
> >
> > Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

4.2.2 Agent Functions

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

> Provides a shortcut of `Admin.query()` that fetches various agent information.

> **Note:** All methods in this function class require your API access key to have the *admin* privilege.

> **classmethod async for ... in paginated_list**()
>
> > classmethod(function) -> method
> >
> > Convert a function to be a class method.
> >
> > A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):
>
> > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await detail()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):
>
> > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

4.2.3 Auth Functions

class ai.backend.client.func.auth.**Auth**
Provides the function interface for login session management and authorization.

classmethod await login()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):
>
> > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await logout()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await update_password()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

4.2.4 Configuration

ai.backend.client.config.**get_env**(*key*, *default=<object object>*, **,* *clean=<function <lambda>>*)
Retrieves a configuration value from the environment variables. The given *key* is uppercased and prefixed by "BACKEND_" and then "SORNA_" if the former does not exist.

Parameters

- **key** (str) – The key name.

- **default** (`Any`) – The default value returned when there is no corresponding environment variable.

- **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.

> **Returns** The value processed by the *clean* function.

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*)

> Sets the configuration used throughout the current process.

class ai.backend.client.config.**APIConfig**(**, endpoint=None, endpoint_type=None, domain=None, group=None, version=None, user_agent=None, access_key=None, secret_key=None, hash_type=None, vfolder_mounts=None, skip_sslcert_validation=None, connection_timeout=None, read_timeout=None, announcement_handler=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.

> **Parameters**

> - **endpoint** (`Union`[`str`, `URL`, None]) – The URL prefix to make API requests via HTTP/HTTPS. If this is given as `str` and contains multiple URLs separated by comma, the underlying HTTP request-response facility will perform client-side load balancing and automatic failover using them, assuming that all those URLs indicates a single, same cluster. The users of the API and CLI will get network connection errors only when all of the given endpoints fail – intermittent failures of a subset of endpoints will be hidden with a little increased latency.

> - **endpoint_type** (`Optional`[`str`]) – Either `"api"` or `"session"`. If the endpoint type is `"api"` (the default if unspecified), it uses the access key and secret key in the configuration to access the manager

API server directly. If the endpoint type is `"session"`, it assumes the endpoint is a Backend.AI console server which provides cookie-based authentication with username and password. In the latter, users need to use `backend.ai login` and `backend.ai logout` to manage their sign-in status, or the API equivalent in `login()` and `logout()` methods.

- **version** (`Optional`[`str`]) – The API protocol version.

- **user_agent** (`Optional`[`str`]) – A custom user-agent string which is sent to the API server as a `User-Agent` HTTP header.

- **access_key** (`Optional`[`str`]) – The API access key. If deliberately set to an empty string, the API requests will be made without signatures (anonymously).

- **secret_key** (`Optional`[`str`]) – The API secret key.

- **hash_type** (`Optional`[`str`]) – The hash type to generate per-request authentication signatures.

- **vfolder_mounts** (`Optional`[`Iterable`[`str`]]) – A list of vfolder names (that must belong to the given access key) to be automatically mounted upon any `Kernel.get_or_create()` calls.

DEFAULTS: Mapping[str, Any] = {'connection_timeout': 10.0, 'domain': 'default', 'end
The default values for config parameterse settable via environment variables xcept the access and secret keys.

endpoint
The currently active endpoint URL. This may change if there are multiple configured endpoints and the current one is not accessible.

> **Return type** URL

endpoints
All configured endpoint URLs.

> **Return type** Sequence[URL]

endpoint_type
The configured endpoint type.

> **Return type** str

domain
The configured domain.

> **Return type** str

group
The configured group.

> > **Return type** str

user_agent

> The configured user agent string.

> > **Return type** str

access_key

> The configured API access key.

> > **Return type** str

secret_key

> The configured API secret key.

> > **Return type** str

version

> The configured API protocol version.

> > **Return type** str

hash_type

> The configured hash algorithm for API authentication signatures.

> > **Return type** str

vfolder_mounts

> The configured auto-mounted vfolder list.

> > **Return type** Sequence[str]

skip_sslcert_validation

> Whether to skip SSL certificate validation for the API gateway.

> > **Return type** bool

connection_timeout

> The maximum allowed duration for making TCP connections to the server.

> > **Return type** float

read_timeout

> The maximum allowed waiting time for the first byte of the response from the server.

> > **Return type** float

announcement_handler

> The announcement handler to display server-set announcements.

> > **Return type** Optional[Callable[[str], None]]

4.2.5 KeyPair Functions

class ai.backend.client.func.keypair.**KeyPair**(*access_key*)

Provides interactions with keypairs.

classmethod await create()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await update()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await delete()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await list()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod async for ... in paginated_list()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

await info(*fields=None*)

Returns the keypair's information such as resource limits.

Parameters fields (Optional[Iterable[str]]) – Additional per-agent query fields to fetch.

New in version 18.12.

> **Return type** dict

classmethod await activate()

> classmethod(function) -> method
>
> Convert a function to be a class method.
>
> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> >
> > > . . .
>
> It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
>
> Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await deactivate()

> classmethod(function) -> method
>
> Convert a function to be a class method.
>
> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> >
> > > . . .
>
> It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
>
> Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

4.2.6 Manager Functions

class ai.backend.client.func.manager.**Manager**

> Provides controlling of the gateway/manager servers.
>
> New in version 18.12.

classmethod await status()

> classmethod(function) -> method
>
> Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod `await freeze`()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod `await unfreeze`()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await get_announcement()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await update_announcement()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await scheduler_op()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

4.2.7 Scaling Group Functions

class ai.backend.client.func.scaling_group.**ScalingGroup**(*name*)

Provides getting scaling-group information required for the current user.

The scaling-group is an opaque server-side configuration which splits the whole cluster into several partitions, so that server administrators can apply different auto-scaling policies and operation standards to each partition of agent sets.

classmethod await list_available()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await list()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await detail()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await create()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await update()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await delete()

> classmethod(function) -> method
>
> Convert a function to be a class method.
>
> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> >
> > > . . .
> >
> > It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
>
> Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await associate_domain()

> classmethod(function) -> method
>
> Convert a function to be a class method.
>
> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> >
> > > . . .
> >
> > It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
>
> Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await dissociate_domain()

> classmethod(function) -> method
>
> Convert a function to be a class method.
>
> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> >
> > > . . .
> >
> > It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await dissociate_all_domain()
    classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

    class C: @classmethod def f(cls, arg1, arg2, . . .):

        . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await associate_group()
    classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

    class C: @classmethod def f(cls, arg1, arg2, . . .):

        . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await dissociate_group()
    classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

    class C: @classmethod def f(cls, arg1, arg2, . . .):

        . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await dissociate_all_group()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):
>
> > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

4.2.8 ComputeSession Functions

class ai.backend.client.func.session.**ComputeSession**(*name,*

owner_access_key=None)

Provides various interactions with compute sessions in Backend.AI.

The term 'kernel' is now deprecated and we prefer 'compute sessions'. However, for historical reasons and to avoid confusion with client sessions, we keep the backward compatibility with the naming of this API function class.

For multi-container sessions, all methods take effects to the master container only, except *destroy()* and *restart()* methods. So it is the user's responsibility to distribute uploaded files to multiple containers using explicit copies or virtual folders which are commonly mounted to all containers belonging to the same compute session.

classmethod async for ... in paginated_list()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await hello()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod async for ... in get_task_logs()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await get_or_create()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await `create_from_template`()
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

await `destroy`(**, forced=False* **)**
Destroys the compute session. Since the server literally kills the container(s), all ongoing executions are forcibly interrupted.

await `restart`()
Restarts the compute session. The server force-destroys the current running container(s), but keeps their temporary scratch directories intact.

await `interrupt`()
Tries to interrupt the current ongoing code execution. This may fail without any explicit errors depending on the code being executed.

await `complete`(*code, opts=None* **)**
Gets the auto-completion candidates from the given code string, as if a user has pressed the tab key just after the code in IDEs.

Depending on the language of the compute session, this feature may not be supported. Unsupported sessions returns an empty list.

Parameters

- **code** (str) – An (incomplete) code text.

- **opts** (Optional[dict]) – Additional information about the current cursor position, such as row, col, line and the remainder text.

Return type Iterable[str]

Returns An ordered list of strings.

await get_info()

Retrieves a brief information about the compute session.

await get_logs()

Retrieves the console log of the compute session container.

await execute(*run_id=None*, *code=None*, *mode='query'*, *opts=None*)

Executes a code snippet directly in the compute session or sends a set of build/clean/execute commands to the compute session.

For more details about using this API, please refer the official API documentation.

Parameters

- **run_id** (Optional[str]) – A unique identifier for a particular run loop. In the first call, it may be None so that the server auto-assigns one. Subsequent calls must use the returned runId value to request continuation or to send user inputs.

- **code** (Optional[str]) – A code snippet as string. In the continuation requests, it must be an empty string. When sending user inputs, this is where the user input string is stored.

- **mode** (str) – A constant string which is one of "query", "batch", "continue", and "user-input".

- **opts** (Optional[dict]) – A dict for specifying additional options. Mainly used in the batch mode to specify build/clean/execution commands. See the API object reference for details.

Returns An execution result object

await upload(*files*, *basedir=None*, *show_progress=False*)

Uploads the given list of files to the compute session. You may refer them in the batch-mode execution or from the code executed in the server afterwards.

Parameters

- **files** (Sequence[Union[str, Path]]) – The list of file paths in the client-side. If the paths include directories, the location of them in

the compute session is calculated from the relative path to *basedir* and all intermediate parent directories are automatically created if not exists.

For example, if a file path is `/home/user/test/data.txt` (or `test/data.txt`) where *basedir* is `/home/user` (or the current working directory is `/home/user`), the uploaded file is located at `/home/work/test/data.txt` in the compute session container.

- **basedir** (Union[str, Path, None]) – The directory prefix where the files reside. The default value is the current working directory.

- **show_progress** (bool) – Displays a progress bar during uploads.

await download(*files, dest='.', show_progress=False*)

Downloads the given list of files from the compute session.

> **Parameters**
>
> - **files** (Sequence[Union[str, Path]]) – The list of file paths in the compute session. If they are relative paths, the path is calculated from `/home/work` in the compute session container.
>
> - **dest** (Union[str, Path]) – The destination directory in the client-side.
>
> - **show_progress** (bool) – Displays a progress bar during downloads.

await list_files(*path='.'*)

Gets the list of files in the given path inside the compute session container.

> **Parameters path** (Union[str, Path]) – The directory path in the compute session.

listen_events()

Opens the stream of the kernel lifecycle events. Only the master kernel of each session is monitored.

> **Return type** SSEContextManager
>
> **Returns** a StreamEvents object.

stream_events()

Opens the stream of the kernel lifecycle events. Only the master kernel of each session is monitored.

> **Return type** SSEContextManager
>
> **Returns** a StreamEvents object.

stream_pty()

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

> > **Return type** *WebSocketContextManager*
>
> > **Returns** 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.
>
> > **Return type** *WebSocketContextManager*

class ai.backend.client.func.session.**StreamPty**(*session, underlying_response, **kwargs*)

> A derivative class of *WebSocketResponse* which provides additional functions to control the terminal.

4.2.9 Session Template Functions

class ai.backend.client.func.session_template.**SessionTemplate**(*template_id, owner_access_key=None*)

> **classmethod await create**()
>
> > classmethod(function) -> method
>
> > Convert a function to be a class method.
>
> > A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> > >
> > > > . . .
>
> > It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.
>
> > Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.
>
> **classmethod await list_templates**()
>
> > classmethod(function) -> method
>
> > Convert a function to be a class method.
>
> > A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:
>
> > > **class C:** @classmethod def f(cls, arg1, arg2, . . .):
> > >
> > > > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

await get(*body_format='yaml'*)

> **Return type** str

await put(*template*)

> **Return type** Any

await delete()

> **Return type** Any

4.2.10 Virtual Folder Functions

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

classmethod await create()

> classmethod(function) -> method

> Convert a function to be a class method.

> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):

> > > . . .

> It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

> Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await delete_by_id()

> classmethod(function) -> method

> Convert a function to be a class method.

> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> > **class C:** @classmethod def f(cls, arg1, arg2, . . .):

> . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await `list`()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):
>
> > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod async for ... in `paginated_list`()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

> **class C:** @classmethod def f(cls, arg1, arg2, . . .):
>
> > . . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await `list_hosts`()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await list_all_hosts()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await list_allowed_types()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

await info()

await delete()

await rename(*new_name*)

await upload(*files, basedir=None, show_progress=False*)

await mkdir(*path*)

await request_download(*filename*)

await rename_file(*target_path, new_name*)

await delete_files(*files, recursive=False*)

await download(*files, show_progress=False*)

await list_files(*path='.'*)

await invite(*perm, emails*)

classmethod await invitations()
> classmethod(function) -> method

> Convert a function to be a class method.

> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

>> **class C:** @classmethod def f(cls, arg1, arg2, . . .):

>>> . . .

> It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

> Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await accept_invitation()
> classmethod(function) -> method

> Convert a function to be a class method.

> A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

>> **class C:** @classmethod def f(cls, arg1, arg2, . . .):

>>> . . .

> It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await delete_invitation()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await get_fstab_contents()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await list_mounts()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await mount_host()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

classmethod await umount_host()

classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, . . .):

. . .

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

await leave()

4.3 Low-level SDK Reference

4.3.1 Base Function

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.

4.3.2 Request 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*, *override_api_version=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:

```python
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())
```

```python
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())
```

>> **Return type** *FetchContextManager*

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

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

> **Return type** *WebSocketContextManager*

content
    Retrieves the content in the original form. Private codes should NOT use this as
    it incurs duplicate encoding/decoding.

> **Return type** Union[bytes, bytearray, str, StreamReader, IOBase,
>   None]

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

> **Return type** None

set‿json(*value*)
    A shortcut for set_content() with JSON objects.

> **Return type** None

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

> **Return type** None

connect‿events(***kwargs*)
    Creates a Server-Sent Events connection.

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

> **Return type** SSEContextManager

class ai.backend.client.request.**Response**(*session,   underlying_response,   *,*
                                async_mode=False, **kwargs)

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

---

class ai.backend.client.request.**FetchContextManager**(*session,*
rqst_ctx_builder, **,*
response_cls=<class
'ai.backend.client.request.Response'>,
check_status=True)

    The context manager returned by *Request.fetch()*.

    It provides both synchronous and asynchronous context manager interfaces.

class ai.backend.client.request.**WebSocketContextManager**(*session,*
ws_ctx_builder,
*, on_enter=None,
re-
sponse_cls=<class
'ai.backend.client.request.WebSocketF

    The context manager returned by *Request.connect_websocket()*.

class ai.backend.client.request.**AttachedFile**(*filename,* *stream,* *con-*
tent_type)

    A struct that represents an attached file to the API request.

        Parameters

- **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".

    content_type

        Alias for field number 2

    count()

        Return number of occurrences of value.

    filename

        Alias for field number 0

    index()

        Return first index of value.

        Raises ValueError if the value is not present.

    stream

        Alias for field number 1

4.3.3 Exceptions

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 and connection
>   failures.

>   **with_traceback**()

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

4.3.4 Miscellaneous Utilities

ai.backend.client.utils.**undefined**

>   A placeholder to signify an undefined value as a singleton object of Undefined and
>   distinguish it from a null (None) value.

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a