neuro Sdk Documentation

Release 22.7.1

Neu.ro Inc.

Feb 20, 2023

CONTENTS

1 Installation **3**

2 Getting Started **5**

3 Contents **7**

3.1 Usage 7

3.2 Reference 11

4 Indices and tables **65**

Python Module Index **67**

Index **69**

A Python library for the Neuro Platform API.

CHAPTER

ONE

INSTALLATION

The latest stable release is available on PyPI. Either add `neuro-sdk` to your `requirements.txt` or install with pip:

```
$ pip install -U neuro-sdk
```

CHAPTER

TWO

GETTING STARTED

To start working with the Neuro Platform you need to login first. The easiest way to do it is the using of *CLI* utility:

```
$ neuro login
```

After the login a configuration file is created and it can be read later.

Use *neuro_sdk.get()* for initializing client instance from existing configuration file:

```
import neuro_sdk

async with neuro_sdk.get() as client:
    async with client.jobs.list() as job_iter:
        jobs = [job async for job in job_iter]
```

The example above instantiates a `client` object in *async context manager* and fetches a list of user's jobs. On exit from `async with` statement the `client` object is closed and is not available for future calls.

See *Usage* section for ideas how typical operations can be done with Neu.ro platform. *Reference* section contains the full API reference for all API classes, functions etc.

CHAPTER

THREE

CONTENTS

3.1 Usage

3.1.1 Jobs Usage

Use Jobs API (available as *Client.jobs*) for starting a job, killing it, getting list of running jobs etc. This chapter describes several common scenarios.

Here we describe the most common scenarios, see *Jobs API Reference* for the full list of job namespace methods.

Start a Job

To start a job use *Jobs.start()* method.

The method accepts image and required resources preset name as parameters and returns *JobDescription* with information about started job:

```
from neuro_sdk import get

async with get() as client:
    job = await client.jobs.start(
        image=client.parse.remote_image("ubuntu:latest"),
        preset_name="cpu-small",
        command="sleep 30",
    )
```

The example above starts a job using `ubuntu:latest` public image, `cpu-small` resources preset and executes `sleep 30` command inside started container.

Note: After return from the call a new job is *scheduled* for execution but usually it's status is *pending*. The Neuro Platform takes time for preparing resources for a job, pulling image from registry etc. Startup time can vary from seconds for *hot start* to minutes for *cold* one.

Check Job Status

After spawning a job we have *JobDescription* instance that describes job status (and other things like executed command line).

A job takes time for deployment, it can be terminated by different reasons, e.g. requested image name doesn't exist.

The following snippet waits for job's *starting execution* or *failure*:

```
# job is a JobDescription given by former client.job.run() call

while True:
    job = await client.jobs.status(job.id)
    if job.status in (JobStatus.RUNNING, JobStatus.SUCCEEDED):
        break
    elif job.status == JobStatus.FAILED:
        raise RuntimeError(f"Job {job.id} failed with {job.reason}:"
                           f"{job.history.description}")
    else:
        await asyncio.sleep(1)
```

Mount Neuro Storage folders

The Neuro Platform provides access to Neuro storage (`storage://`) by *mounted folders* inside a container (*volumes* in Docker glossary).

If you have a directory `storage:folder` and want to mount it inside a container under `/var/data` path please create a *Volume* and use it in *Container* definition:

```
from yarl import URL

volume = Volume(
    storage_uri=URL("storage:folder"),
    container_path="/mnt/data",
)

job = await client.jobs.run(
    Container(
        image=client.parse.remote_image("ubuntu:latest"),
        resources=Resources(memory_mb=100, cpu=0.5),
        command="sleep 30",
        volumes=[volume],
    )
)
```

There is a parsing helper that can construct a *Volume* instance from a string in format that is used in *CLI*:

```
volume = client.parse.volume("storage:folder:/var/data")
```

To specify *read-only* mount point please pass `read_only=True` to *Volume* constructor, e.g. the following code mounts public shared `storage://neuro/public` folder in read-only mode:

```
public_volume = Volume(
    storage_uri=URL("storage:neuro/public"),
```

(continues on next page)

(continued from previous page)

```
    container_path="/mnt/public",
    read_only=True,
)
```

The same effect can be achieved by using a parser API:

```
public_volume = client.parse.volume(
    "storage:neuro/public:/mnt/public:ro")
```

Pass a list of *volumes* into container to support multiple mount points:

```
Container(
    image=...,
    resources=...,
    command=...,
    volumes=[volume, public_volume],
)
```

See also:

Storage Usage for the storage manipulation API.

Kill a Job

Use *Jobs.kill()* for enforcing job to stop:

```
await client.jobs.kill(job.id)
```

Expose job's TCP ports locally

Sometimes there is a need to access TCP endpoints exposed by a job executed on the Neuro Platform from local workstation.

For example, you've started a gRPC server inside a container on TCP port `12345` and want to access this service from your laptop.

You need to bridge this *remote* `12345` port into a local TCP namespace (e.g. `23456` *local* port by *Jobs.port_forward()* method:

```
from grpclib.client import Channel

# generated by protoc
from .helloworld_pb2 import HelloRequest, HelloReply
from .helloworld_grpc import GreeterStub

 async with client.jobs.port_forward(job.id, 23456, 12345):
     # open gRPC client and use it

     channel = Channel('127.0.0.1', 23456)
     greeter = GreeterStub(channel)

     reply: HelloReply = await greeter.SayHello(HelloRequest(name='Dr. Strange'))
```

(continues on next page)

(continued from previous page)

```
    print(reply.message)

    channel.close()
```

The example uses grpclib library for make client gRPC requests.

Job preemption

Job preemption means that unlike normal jobs preemptible ones can be stopped by kernel when the system has lack of resources and restarted later. All memory and local disk changes are lost but data written to the Neuro Storage (see *Mount Neuro Storage folders*) is persistent.

To support preemption job's code should be organized in the following way: it dumps *snapshots* on disk periodically. On restart the code checks for last saved snapshot and continues the work from this point.

AI frameworks usually supports snapshots out of the box, see saving and loading models in pytorch or Keras ModelCheckpoint.

Preemptible job is not such convenient as regular job but it's computational time is much cheaper (exact numbers varies on concrete computational cluster provides, e.g. Google Compute, AWS or Azure).

Jobs are *non-preeptible* by default, you can change this by passing `preemptible_node=True` flag to *`Jobs.run()`*.

3.1.2 Storage Usage

Use Storage API (available as *`Client.storage`*) for uploading files to the Neu.ro Storage and downloading them back. This chapter describes several common scenarios like uploading / downloading directories recursively.

There are many methods in *`Storage`* namespace, here we describe a few.

Blob Storage API (available as `Client.blob_storage`) is another subsystem, which has a similar Upload/Download interface as methods shown below. Please refer to `BlobStorage` documentation for more details.

Upload a Folder

Use *`Storage.upload_dir()`* to upload a local directory on the Neuro Storage:

```
from neuro_sdk import get
from yarl import URL

async with get() as client:
    await client.storage.upload_dir(
        URL("file:local_folder"),
        URL("storage:remote_folder"),
    )
```

The example above recursively uploads all files and directories `./local_folder` to `storage://<username>/remote_folder`.

Use `update=True` flag to upload only files that are newer than are present on the Storage:

```
await client.storage.upload_dir(
    URL("file:local_folder"),
    URL("storage:remote_folder"),
    update=True,
)
```

Download a Folder

Use *Storage.download_dir()* for downloading data from the Neuro Storage to local disk.

The method is a counterpart to *Storage.upload_dir()* and has the same arguments:

```
await client.storage.download_dir(
    URL("storage:remote_folder"),
    URL("file:local_folder"),
)
```

The example above recursively downloads files and directories from `storage:remote_folder` to `./local_folder`.

3.2 Reference

3.2.1 Initialization

API functions

async-with neuro_sdk.**get**(*, *path: Optional[Path] = None*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → AsyncContextManager[*Client*]

The handy API for getting initialized *Client* instance.

A shortcut for *Factory.get()* that acts as asynchronous context manager.

The usage is:

```
async with neuro_sdk.get() as client:
    async with client.jobs.list() as jobs:
        async for job in jobs:
            print(job.id)
```

See *Factory.get()* for optional function arguments meaning.

coroutine neuro_sdk.**login**(*show_browser_cb: Callable[[URL], Awaitable[None]]*, *, *url: URL = DEFAULT_API_URL*, *path: Optional[Path] = None*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

A shortcut for *Factory.login()*. See the method for details.

coroutine neuro_sdk.**login_with_headless**(*get_auth_code_cb: Callable[[URL], Awaitable[str]]*, *, *url: URL = DEFAULT_API_URL*, *path: Optional[Path] = None*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

A shortcut for *Factory.login_headless()*. See the method for details.

coroutine neuro_sdk.**login_with_token**(*token: str*, *, *url: URL = DEFAULT_API_URL*, *path: Optional[Path] = None*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

A shortcut for *Factory.login_with_token()*. See the method for details.

coroutine neuro_sdk.**logout**(*, *path: Optional[Path] = None*, *show_browser_cb: Callable[[URL], Awaitable[None]] = None*) → None

A shortcut for *Factory.logout()*. See the method for details.

Config Factory

class neuro_sdk.**Factory**(*path: Optional[Path]*)

A *factory* that used for making *Client* instances, logging into Neuro Platform and logging out.

path (pathlib.Path) can be provided for pointing on a *custom* configuration directory (~/.nmrc by default). The default value can be overridden by NEUROMATION_CONFIG environment variable.

path

Revealed path to the configuration directory, expanded as described above.

Read-only pathlib.Path property.

New in version 20.2.25.

is_config_present

True if config files are present under *path*, False otherwise.

Read-only bool property.

coroutine get(*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → *Client*

Read configuration previously created by *login methods* and return a client instance. Update authorization token if needed.

The easiest way to create required configuration file is running `neuro login` *CLI* command before the first call of this method from a user code.

Parameters
timeout (*aiohttp.ClientTimeout*) – optional timeout for HTTP operations, see also *Timeouts*.

Returns
Client that can be used for working with Neuro Platform.

Raise
ConfigError if configuration file doesn't exist, malformed or not compatible with SDK version.

coroutine login(*show_browser_cb: Callable[[URL], Awaitable[None]]*, *, *url: URL = DEFAULT_API_URL*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

Log into Neuro Platform using in-browser authorization method.

The method is dedicated for login from workstation with GUI system. For logging in from server please use *login_headless()*.

The caller provides *show_browser_cb* callback which is called with URL argument.

The callback should open a browser with this URL (webbrowser.open() can be used).

After the call the configuration file is created, call *get()* for making a client and performing Neuro Platform operations.

Parameters

- **show_browser_cb** – a callback that should open a browser with specified URL for handling authorization.
- **url** (*URL*) – Neuro Platform API URL, `URL("https://staging.neu.ro/api/v1")` by default.
- **timeout** (*aiohttp.ClientTimeout*) – optional timeout for HTTP operations, see also *Timeouts*.

coroutine login_headless(*get_auth_code_cb: Callable[[URL], Awaitable[str]]*, *, *url: URL = DEFAULT_API_URL*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

Log into Neuro Platform using two-step authorization method.

The method is dedicated for login from remote server that has no GUI system. For logging in from GUI equipped workstation please use *login()*.

The caller provides *get_auth_code_cb* callback which is called with URL argument.

Usually, the callback prints given URL on screen and displays a prompt.

User copies the URL from remote terminal session into local browser, authorizes and enters authorization code shown in the browser back into prompt.

After the call the configuration file is created, call *get()* for making a client and performing Neuro Platform operations.

Parameters

- **get_auth_code_cb** – a callback that receives an URL and returns authorization code.
- **url** (*URL*) – Neuro Platform API URL, `URL("https://staging.neu.ro/api/v1")` by default.
- **timeout** (*aiohttp.ClientTimeout*) – optional timeout for HTTP operations, see also *Timeouts*.

coroutine login_with_token(*token: str*, *, *url: URL = DEFAULT_API_URL*, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

Log into Neuro Platform using previously acquired token. The method is deprecated and not recommended to use. Provided tokens will be revoked eventually.

Parameters

- **token** (*str*) – authorization token.
- **url** (*URL*) – Neuro Platform API URL, `URL("https://staging.neu.ro/api/v1")` by default.
- **timeout** (*aiohttp.ClientTimeout*) – optional timeout for HTTP operations, see also *Timeouts*.

coroutine login_with_passed_config(*config_data: Optional[str] = None*, *, *timeout: aiohttp.ClientTimeout = DEFAULT_TIMEOUT*) → None

Log into Neuro Platform using config data passed by platform. Use this only to login from the job that was started with `pass_config=True`. Inside such job, *config_data* is available under `NEURO_PASSED_CONFIG` environment variable.

Parameters

- **config_data** (*str*) – config data passed by platform.

- **timeout** (*aiohttp.ClientTimeout*) – optional timeout for HTTP operations, see also *Timeouts*.

coroutine logout(*show_browser_cb: Callable[[URL], Awaitable[None]] = None*)

Log out from Neuro Platform. In case *show_browser_cb* callback passed, the browser will be opened to remove session cookie.

Parameters

show_browser_cb – a callback that should open a browser with specified URL for handling authorization.

Timeouts

By default the SDK raises `asyncio.TimeoutError` if the server doesn't respond in a minute. It can be overridden by passing *timeout* argument to *Factory* methods.

3.2.2 Client class

class neuro_sdk.**Client**

Neuro Platform client.

For creating a client instance use *Factory* or *get()*.

The class provides access to Neu.ro subsystems like *jobs* or *storage*.

username

User name used for working with Neuro Platform, read-only `str`.

presets

A `typing.Mapping` of preset name (`str`) to *Preset* dataclass.

Presets are loaded from server on login.

config

Configuration subsystem, see *Config* for details.

jobs

Jobs subsystem, see *Jobs* for details.

storage

Storage subsystem, see *Storage* for details.

users

Users subsystem, see *Users* for details.

images

Images subsystem, see *Images* for details.

secrets

Images subsystem, see *Secrets* for details.

disks

Images subsystem, see *Disks* for details.

parse

A set or helpers used for parsing different Neuro API definitions, see *Parser* for details.

coroutine close()

Close Neuro API client, all calls after closing are forbidden.

The method is idempotent.

closed

Is this client instance was closed, bool.

3.2.3 Configuration API Reference

API functions

neuro_sdk.**find_project_root**(*path: Optional[Path] = None*) → Path

Look for project root directory.

Folder is considered a project root when it contains `.neuro.toml`. Search begins at **path** or `Path.cwd()` (current working directory) when **path** is **None** and checks all parent folders sequentially. Will raise an `ConfigError` if search reaches root directory.

Config

class neuro_sdk.**Config**

Configuration subsystem, available as *Client.config*.

Use it for analyzing fetching information about the system configuration, e.g. a list of available clusters or switching the active cluster.

username

User name used for working with Neuro Platform, read-only str.

presets

A typing.Mapping of preset name (str) to *Preset* dataclass for the current cluster.

clusters

A typing.Mapping of cluster name (str) to *Cluster* dataclass for available clusters.

cluster_name

The current cluster name, read-only str.

To switch on another cluster use *switch_cluster()*.

org_name

The current org name, read-only str.

To switch on another org use *switch_org()*.

coroutine fetch() → None

Fetch available clusters configuration from the Neuro Platform.

Note: The call updates local configuration files.

coroutine switch_cluster(*name: str*) → None

Switch the current cluster to *name*.

Note: The call updates local configuration files.

coroutine switch_org(*name: str*) → None

Switch the current org to *name*.

Note: The call updates local configuration files.

Miscellaneous helpers

api_url

The Neuro Platform URL, `yarl.URL`.

registry_url

Docker Registry URL for the cluster, `yarl.URL`.

Cluster.registry_url for the current cluster.

storage_url

Storage URL for the cluster, `yarl.URL`.

Cluster.storage_url for the current cluster.

monitoring_url

Monitoring URL for the cluster, `yarl.URL`.

Cluster.monitoring_url for the current cluster.

coroutine get_user_config() → Mapping[str, Any]

Return user-provided config dictionary. Config is loaded from config files. There are two configuration files: **global** and **local**, both are optional and can be absent.

The global file is named `user.toml` and the API search for it in the path provided to *Factory* or *get()* (`$HOME/.neuro/user.toml` by default).

The local config file is named `.neuro.toml`, and the API search for this file starting from the current folder up to the root directory.

Found local and global configurations are merged. If a parameter is present are both global and local versions the local parameter take a precedence.

Configuration files have a TOML format (a stricter version of well-known INI format). See https://en.wikipedia.org/wiki/TOML and https://github.com/toml-lang/toml#toml for the format specification details.

The API will raise an `ConfigError` if configuration files contains unknown sections or parameters. Note that currently API doesn't use any parameter from user config.

Known sections: **alias**, **job**, **storage**.

Section **alias** can have any subsections with any keys.

Section **job** can have following keys: **ps-format** - string, **life-span** - string.

Section **storage** can have following keys: **cp-exclude** - list of strings, **cp-exclude-from-files** - list of strings.

There is a plugin system that allows to register additional config parameters. To define a plugin, add a **neuro_api** entrypoint (check https://packaging.python.org/specifications/entry-points/ for more info about entry points). Entry point should be callable with single argument of type *PluginManager*.

New in version 20.01.15.

coroutine token() → str

Bearer token to log into the Neuro Platform.

The token expires after some period, the call automatically refreshes the token if needed.

Cluster

class neuro_sdk.**Cluster**

Read-only `dataclass` for describing a cluster configuration.

Clusters are loaded on login to the Neuro platform and updated on *Config.fetch()* call.

Config.switch_cluster() changes the active cluster.

name

Cluster name, `str`.

registry_url

Docker Registry URL for the cluster, `yarl.URL`.

storage_url

Storage URL for the cluster, `yarl.URL`.

users_url

Users URL for the cluster, `yarl.URL`.

monitoring_url

Monitoring URL for the cluster, `yarl.URL`.

presets

A `typing.Mapping` of available job resource presets, keys are preset names (`str`), values are *Preset* objects.

Preset

class neuro_sdk.**Preset**

Read-only `dataclass` for describing a job configuration provided by Neuro Platform.

Presets list is loaded on login to the Neuro platform and depends on used cluster.

cpu

Requested number of CPUs, `float`. Please note, Docker supports fractions here, e.g. `0.5` CPU means a half or CPU on the target node.

memory_mb

Requested memory amount in MegaBytes, `int`.

is_preemptible

A flag that specifies is the job is *preemptible* or not, see *Preemption* for details.

gpu

The number of requested GPUs, `int`. Use `None` for jobs that doesn't require GPU.

gpu_model

The name of requested GPU model, `str` (or `None` for job without GPUs).

tpu_type

Requested TPU type, see also https://en.wikipedia.org/wiki/Tensor_processing_unit

tpu_software_version

Requested TPU software version.

3.2.4 Jobs API Reference

Jobs

class neuro_sdk.**Jobs**

Jobs subsystem, available as *Client.jobs*.

User can start new job, terminate it, get status, list running jobs etc.

async-with attach(*id: str*, **, tty: bool = False, stdin: bool = False, stdout: bool = False, stderr: bool = False, cluster_name: Optional[str] = None*) → AsyncContextManager[*StdStream*]

Get access to standard input, output, and error streams of a running job.

Parameters

- **id** (*str*) – job *id* to use for command execution.
- **tty** (*bool*) – `True` if *tty* mode is requested, default is `False`.
- **stdin** (*bool*) – `True` to attach stdin, default is `False`.
- **stdout** (*bool*) – `True` to attach stdout, default is `False`.
- **stderr** (*bool*) – `True` to attach stderr, default is `False`.
- **cluster_name** (*str*) – cluster on which the job is running.

  `None` means the current cluster (default).

Returns

Asynchronous context manager which can be used to access stdin/stdout/stderr, see *StdStream* for details.

async-with exec(*id: str*, *cmd: str*, **, tty: bool = False, stdin: bool = False, stdout: bool = False, stderr: bool = False, cluster_name: Optional[str] = None*) → AsyncContextManager[*StdStream*]

Start an exec session, get access to session's standard input, output, and error streams.

Parameters

- **id** (*str*) – job *id* to use for command execution.
- **cmd** (*str*) – the command to execute.
- **tty** (*bool*) – `True` if *tty* mode is requested, default is `False`.
- **stdin** (*bool*) – `True` to attach stdin, default is `False`.
- **stdout** (*bool*) – `True` to attach stdout, default is `False`.
- **stderr** (*bool*) – `True` to attach stderr, default is `False`.

- **cluster_name** (*str*) – cluster on which the job is running.

  None means the current cluster (default).

Returns
Asynchronous context manager which can be used to access stdin/stdout/stderr, see *StdStream* for details.

coroutine get_capacity(*, *cluster_name: Optional[str] = None*) → Mapping[str, int]

Get counts of available job for specified cluster for each available preset.

The returned numbers reflect the remaining *cluster capacity*. In other words, it displays how many concurrent jobs for each preset can be started at the moment of the method call.

The returned capacity is an approximation, the real value can differ if already running jobs are finished or another user starts own jobs at the same time.

Parameters
cluster_name (*str*) – cluster for which the request is performed.

None means the current cluster (default).

Returns
A mapping of *preset_name* to *count*, where *count* is a number of concurrent jobs that can be executed using *preset_name*.

coroutine kill(*id: str*) → None

Kill a job.

Parameters
id (*str*) – job *id* to kill.

async-with async-for list(*, *statuses: Iterable[JobStatus] = ()*, *name: Optional[str] = None*, *tags: Sequence[str] = ()*, *owners: Iterable[str] = ()*, *since: Optional[datetime] = None*, *until: Optional[datetime] = None*, *reverse: bool = False*, *limit: Optional[int] = None*, *cluster_name: Optional[str] = None*) → AsyncContextManager[AsyncIterator[*JobDescription*]]

List user jobs, all scheduled, running and finished jobs by default.

Parameters

- **statuses** (*Iterable[JobStatus]*) – filter jobs by their statuses.

  The parameter can be a set or list of requested statuses, e.g. {JobStatus.PENDIND, JobStatus.RUNNING} can be used for requesting only scheduled and running job but skip finished and failed ones.

  Empty sequence means that jobs with all statuses are returned (default behavior). The list can be pretty huge though.

- **name** (*str*) – Filter jobs by *name* (exact match).

  Empty string or None means that no filter is applied (default).

- **tags** (*Sequence[str]*) – filter jobs by *tags*.

  Retrieves only jobs submitted with all tags from the specified list.

  Empty list means that no filter is applied (default).

- **owners** (*Iterable[str]*) – filter jobs by their owners.

  The parameter can be a set or list of owner usernames (see *JobDescription.owner* for details).

No owners filter is applied if the iterable is empty.

- **since** (*datetime*) – filter jobs by their creation date.

  Retrieves only jobs submitted after the specified date (including) if it is not None. If the parameter is a naive datetime object, it represents local time.

  None means that no filter is applied (default).

- **until** (*datetime*) – filter jobs by their creation date.

  Retrieves only jobs submitted before the specified date (including) if it is not None. If the parameter is a naive datetime object, it represents local time.

  None means that no filter is applied (default).

- **reverse** (*bool*) – iterate jobs in the reverse order.

  If *reverse* is false (default) the jobs are iterated in the order of their creation date, from earlier to later. If *reverse* is true, they are iterated in the reverse order, from later to earlier.

- **limit** (*int*) – limit the number of jobs.

  None means no limit (default).

- **cluster_name** (*str*) – list jobs on specified cluster.

  None means the current cluster (default).

Returns

asynchronous iterator which emits *JobDescription* objects.

```
monitor(id: str, *,                          cluster_name: Optional[str] = None,
since: Optional[datetime] = None,
timestamps: bool = False,
separator: Optional[str] = None,
) -> AsyncContextManager[AsyncIterator[bytes]]
```

Get job logs as a sequence of data chunks, e.g.:

```
async with client.jobs.monitor(job_id) as it:
    async for chunk in it:
        print(chunk.encode('utf8', errors='replace')
```

Parameters

- **id** (*str*) – job *id* to retrieve logs.
- **cluster_name** (*str*) – cluster on which the job is running.

  None means the current cluster (default).

- **since** (*datetime*) – Retrieves only logs after the specified date (including) if it is not None. If the parameter is a naive datetime object, it represents local time.

  None means that no filter is applied (default).

- **timestamps** (*bool*) – if true, include timestamps on each line in the log output.
- **separator** (*str*) – string which will separate archive and live logs (if both parts are present).

  By default a string containing random characters are used. Empty *separator* suppresses output of separator.

Returns
AsyncIterator over bytes log chunks.

async-with port_forward(*id: str*, *local_port: int*, *job_port: int*, ***, *no_key_check: bool = False*, *cluster_name: Optional[str] = None*) → None

Forward local port to job, e.g.:

```
async with client.jobs.port_forward(job_id, 8080, 80):
    # port forwarding is awailable inside with-block
```

Parameters

- **id** (*str*) – job *id*.
- **local_port** (*int*) – local TCP port to forward.
- **jot_port** (*int*) – remote TCP port in a job to forward.
- **cluster_name** (*str*) – cluster on which the job is running.

  None means the current cluster (default).

coroutine run(*container: Container*, ***, *name: Optional[str] = None*, *tags: Sequence[str] = ()*, *description: Optional[str] = None*, *scheduler_enabled: bool = False*, *pass_config: bool = False*, *wait_for_jobs_quota: bool = False*, *schedule_timeout: Optional[float] = None*, *life_span: Optional[float] = None*, *priority: Optional[JobPriority] = None*) → *JobDescription*

Start a new job.

Deprecated since version 20.11.25: Please use *start()* instead.

Parameters

- **container** (Container) – container description to start.
- **name** (*str*) – optional container name.
- **name** – optional job tags.
- **description** (*str*) – optional container description.
- **scheduler_enabled** (*bool*) – a flag that specifies is the job should participate in round-robin scheduling.
- **pass_config** (*bool*) – a flag that specifies that platform should pass config data to job. This allows to API and CLI from the inside of the job. See *Factory.login_with_passed_config()* for details.
- **wait_for_jobs_quota** (*bool*) – when this flag is set, job will wait for another job to stop instead of failing immediately because of total running jobs quota.
- **schedule_timeout** (*float*) – minimal timeout to wait before reporting that job cannot be scheduled because the lack of computation cluster resources (memory, CPU/GPU etc). This option is not allowed when `is_preemptible` is set to `True`.
- **life_span** (*float*) – job run-time limit in seconds. Pass *None* to disable.
- **priority** (JobPriority) – priority used to specify job's start order. Jobs with higher priority will start before ones with lower priority. Priority should be supported by cluster.

Returns
JobDescription instance with information about started job.

coroutine **start**(*, *image:* RemoteImage, *preset_name: str*, *cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*, *entrypoint: Optional[str] = None*, *command: Optional[str] = None*, *working_dir: Optional[str] = None*, *http: Optional[HTTPPort] = None*, *env: Optional[Mapping[str, str]] = None*, *volumes: Sequence[Volume] = ()*, *secret_env: Optional[Mapping[str, URL]] = None*, *secret_files: Sequence[SecretFile] = ()*, *disk_volumes: Sequence[DiskVolume] = ()*, *tty: bool = False*, *shm: bool = False*, *name: Optional[str] = None*, *tags: Sequence[str] = ()*, *description: Optional[str] = None*, *pass_config: bool = False*, *wait_for_jobs_quota: bool = False*, *schedule_timeout: Optional[float] = None*, *restart_policy:* JobRestartPolicy *= JobRestartPolicy.NEVER*, *life_span: Optional[float] = None*, *privileged: bool = False*, *priority: Optional[JobPriority] = None*) → *JobDescription*

Start a new job.

Parameters

- **image** (RemoteImage) – image used for starting a container.
- **preset_name** (*str*) – name of the preset of resources given to a container on a node.
- **cluster_name** (*str*) – cluster to start a job. Default is current cluster.
- **org_name** (*str*) – org to start a job on behalf of. Default is current org.
- **entrypoint** (*str*) – optional Docker ENTRYPOINT used for overriding image entry-point (`str`), default `None` is used to pick entry-point from image's *Dockerfile*.
- **command** (*str*) – optional command line to execute inside a container (`str`), `None` for picking command line from image's *Dockerfile*.
- **working_dir** (*str*) – optional working directory inside a container (`str`), `None` for picking working directory from image's *Dockerfile*.
- **http** (`HTTPPort`) – optional parameters of HTTP server exposed by container, `None` if the container doesn't provide HTTP access.
- **env** (*Mapping[str, str]*) – optional custom *environment variables* for pushing into container's task. A `Mapping` where keys are environments variables names and values are variable values, both `str`. `None` by default.
- **volumes** (*Sequence[Volume]*) – optional Docker volumes to mount into container, a `Sequence` of *Volume* objects. Empty `tuple` by default.
- **secret_env** (*Mapping[str, yarl.URL]*) – optional secrets pushed as custom *environment variables* into container's task. A `Mapping` where keys are environments variables names (`str`) and values are secret URIs (`yarl.URL`). `None` by default.
- **secret_files** (*Sequence[SecretFile]*) – optional secrets mounted as files in a container, a `Sequence` of *SecretFile* objects. Empty `tuple` by default.
- **disk_volumes** (*Sequence[DiskVolume]*) – optional disk volumes used to mount into container, a `Sequence` of *DiskVolume* objects. Empty `tuple` by default.
- **tty** (*bool*) – Allocate a TTY or not. `False` by default.
- **shm** (*bool*) – Use Linux shared memory or not. `False` by default.
- **name** (*str*) – optional job name.
- **tags** (*Sequence[str]*) – optional job tags.
- **description** (*str*) – optional container description.

- **pass_config** (*bool*) – a flag that specifies that platform should pass config data to job. This allows to API and CLI from the inside of the job. See *Factory.login_with_passed_config()* for details.
- **wait_for_jobs_quota** (*bool*) – when this flag is set, job will wait for another job to stop instead of failing immediately because of total running jobs quota.
- **schedule_timeout** (*float*) – minimal timeout to wait before reporting that job cannot be scheduled because the lack of computation cluster resources (memory, CPU/GPU etc).
- **life_span** (*float*) – job run-time limit in seconds. Pass *None* to disable.
- **restart_policy** (JobRestartPolicy) – job restart behavior. *JobRestartPolicy*.NEVER by default.
- **privileged** (*bool*) – Run job in privileged mode. This mode should be supported by cluster.
- **priority** (JobPriority) – priority used to specify job's start order. Jobs with higher priority will start before ones with lower priority. Priority should be supported by cluster. None by default.

Returns
JobDescription instance with information about started job.

coroutine send_signal(*id: str*, *, *cluster_name: Optional[str] = None*) → None

Send SIGKILL signal to a job.

Parameters

- **id** (*str*) – job *id*.
- **cluster_name** (*str*) – cluster on which the job is running.

  None means the current cluster (default).

coroutine status(*id: str*) → *JobDescription*

Get information about a job.

Parameters
id (*str*) – job *id* to get its status.

Returns
JobDescription instance with job status details.

async-with async-for top(*id: str*, *, *cluster_name: Optional[str] = None*) → AsyncContextManager[AsyncIterator[*JobTelemetry*]]

Get job usage statistics, e.g.:

```
async with client.jobs.top(job_id) as top:
    async for data in top:
        print(data.cpu, data.memory)
```

Parameters

- **id** (*str*) – job *id* to get telemetry data.
- **cluster_name** (*str*) – cluster on which the job is running.

  None means the current cluster (default).

Returns
asynchronous iterator which emits *JobTelemetry* objects periodically.

coroutine bump_life_span(*id: str*, *additional_life_span: float*) → None

Increase life span of a job.

Parameters

- **id** (*str*) – job *id* to increase life span.
- **life_span** (*float*) – amount of seconds to add to job run-time limit.

Container

class neuro_sdk.**Container**

Read-only `dataclass` for describing Docker image and environment to run a job.

image

RemoteImage used for starting a container.

resources

Resources which are used to schedule a container.

entrypoint

Docker ENTRYPOINT used for overriding image entry-point (`str`), default `None` is used to pick entry-point from image's *Dockerfile*.

command

Command line to execute inside a container (`str`), `None` for picking command line from image's *Dockerfile*.

http

HTTPPort for describing parameters of HTTP server exposed by container, `None` if the container doesn't provide HTTP access.

env

Custom *environment variables* for pushing into container's task.

A `Mapping` where keys are environments variables names and values are variable values, both `str`. Empty `dict` by default.

volumes

Docker volumes to mount into container, a `Sequence` of *Volume* objects. Empty `list` by default.

secret_env

Secrets pushed as custom *environment variables* into container's task.

A `Mapping` where keys are environments variables names (`str`) and values are secret URIs (`yarl.URL`). Empty `dict` by default.

secret_files

Secrets mounted as files in a container, a `Sequence` of *SecretFile* objects. Empty `list` by default.

disk_volumes

Disk volumes used to mount into container, a `Sequence` of *DiskVolume* objects. Empty `list` by default.

HTTPPort

class neuro_sdk.**HTTPPort**

Read-only `dataclass` for exposing HTTP server started in a job.

To access this server from remote machine please use *JobDescription.http_url*.

port

Open port number in container's port namespace, `int`.

requires_auth

Authentication in Neuro Platform is required for access to exposed HTTP server if `True`, the port is open publicly otherwise.

JobDescription

class neuro_sdk.**JobDescription**

Read-only `dataclass` for describing a job.

id

Job ID, `str`.

owner

A name of user who created a job, `str`.

cluster_name

A name of cluster where job was scheduled, `str`.

New in version 19.9.11.

status

Current status of job, *JobStatus* enumeration.

history

Additional information about job, e.g. creation time and process exit code. *JobStatusHistory* instance.

container

Description of container information used to start a job, *Container* instance.

scheduler_enabled

Is job participate in round-robin scheduling.

preemptible_node

Is this node allows execution on preemptible node. If set to `True`, the job only allows execution on preemptible nodes. If set to `False`, the job only allows execution on **non**-preemptible nodes.

pass_config

Is config data is passed by platform, see *Factory.login_with_passed_config()* for details.

privileged

Is the job is running in privileged mode, refer to docker documentation for details.

name

Job name provided by user at creation time, `str` or `None` if name is omitted.

tags

List of job tags provided by user at creation time, `Sequence[str]` or `()` if tags omitted.

description

Job description text provided by user at creation time, `str` or `None` if description is omitted.

http_url

`yarl.URL` for HTTP server exposed by job, empty URL if the job doesn't expose HTTP server.

ssh_server

`yarl.URL` to access running job by SSH. Internal field, don't access it from custom code. Use *Jobs.exec()* and *Jobs.port_forward()* as official API for accessing to running job.

internal_hostname

DNS name to access the running job from other jobs.

internal_hostname_named

DNS name to access the running job from other jobs based on jobs name instead of jobs id. Produces same value for jobs with `name` and `owner` in same cluster.

life_span

Job run-time limit in seconds, `float`

schedule_timeout

Minimal timeout in seconds job will wait before reporting it cannot be scheduled because the lack of computation cluster resources (memory, CPU/GPU etc), `float`

priority

Priority used to specify job's start order. Jobs with higher priority will start before ones with lower priority, *JobPriority*

_internal

Some internal info about job used by platform. Should not be used.

JobRestartPolicy

class neuro_sdk.**JobRestartPolicy**

Enumeration that describes job restart behavior.

Can be one of the following statues:

NEVER

Job will never be restarted.

ON_FAILURE

Job will be restarted only in case of job failure.

ALWAYS

Job will always be restarted after success or failure.

JobPriority

class neuro_sdk.**JobPriority**

Enumeration that describes job priority.

Can be one of the following statues:

LOW

Jobs with LOW priority will start after all other jobs.

NORMAL

Default job priority.

HIGH

Jobs with HIGH priority will start before all other jobs.

JobStatus

class neuro_sdk.**JobStatus**

Enumeration that describes job state.

Can be one of the following statues:

PENDING

Job is scheduled for execution but not started yet.

RUNNING

Job is running now.

SUSPENDED

Scheduled job is paused to allow other jobs to run.

SUCCEEDED

Job is finished successfully.

CANCELLED

Job was canceled while it was running.

FAILED

Job execution is failed.

UNKNOWN

Invalid (or unknown) status code, should be never returned from server.

Also some shortcuts are available:

items() → Set[*JobStatus*]

Returns all statuses except *UNKNOWN*.

active_items() → Set[*JobStatus*]

Returns all statuses that are not final: *PENDING*, *SUSPENDED* and *RUNNING*.

finished_items() → Set[*JobStatus*]

Returns all statuses that are final: *SUCCEEDED*, *CANCELLED* and *FAILED*.

Each enum value has next `bool` fields:

is_pending

Job is waiting to become running. `True` for *PENDING* and *SUSPENDED* states.

is_running

Job is running now. `True` for *RUNNING* state.

is_finished

Job completed execution. `True` for *SUCCEEDED*, *CANCELLED* and *FAILED*

JobStatusItem

class neuro_sdk.**JobStatusItem**

Read-only `dataclass` for describing job status transition details.

transition_time

Status transition timestamp, `datetime`.

status

Status of job after this transition, *JobStatus* enumeration.

reason

Additional information for job status, `str`.

Examples of *reason* values:

- `'ContainerCreating'` for *JobStatus.PENDING* job that initiates a pod for container.
- `'ErrImagePull'` for *JobStatus.FAILED* job that cannot pull specified image.

description

Extended description for short abbreviation described by *reason*, empty `str` if no additional information is provided.

exit_code

Exit code for container's process (`int`) or `None` if the job was not started or was still running when this transition occurred.

JobStatusHistory

class neuro_sdk.**JobStatusHistory**

Read-only `dataclass` for describing job status details, e.g. creation and finishing time, exit code etc.

status

Current status of job, *JobStatus* enumeration.

The same as *JobDescription.status*.

reason

Additional information for current status, `str`.

Examples of *reason* values:

- `'ContainerCreating'` for *JobStatus.PENDING* job that initiates a pod for container.
- `'ErrImagePull'` for *JobStatus.FAILED* job that cannot pull specified image.

description

Extended description for short abbreviation described by *reason*, empty `str` if no additional information is provided.

exit_code

Exit code for container's process (`int`) or `None` if the job was not started or is still running.

restarts

Number of container's restarts, `int`.

created_at

Job creation timestamp, `datetime` or `None`.

started_at

Job starting timestamp, `datetime` or `None` if job not started.

finished_at

Job finishing timestamp, `datetime` or `None` if job not finished.

transitions

List of job status transitions, `Sequence` of *JobStatusItem*.

JobTelemetry

class neuro_sdk.**JobTelemetry**

Read-only `dataclass` for job telemetry (statistics), e.g. consumed CPU load, memory footprint etc.

See also:

Jobs.top().

timestamp

Date and time of telemetry report (`float`), time in seconds since the epoch, like the value returned from `time.time()`.

See `time` and `datetime` for more information how to handle the timestamp.

cpu

CPU load, `float`. `1` means fully loaded one CPU unit, `0.5` is for half-utilized CPU.

memory

Consumed memory in megabytes, `float`.

gpu_duty_cycle

Percentage of time over the past sample period (10 seconds) during which the accelerator was actively processing. `int` between `1` and `100`, `None` if the job has no GPU available.

gpu_memory

Percentage of used GPU memory, `float` between `0` and `1`.

Message

class neuro_sdk.**Message**

Read-only `dataclass` for representing job's stdout/stderr stream chunks, returned from *StdStream.read_out()*.

fileno

Stream number, *1* for stdin and *2* for stdout.

data

A chunk of stdout/stderr data, `bytes`.

Resources

class neuro_sdk.**Resources**

Read-only `dataclass` for describing resources (memory, CPU/GPU etc.) available for container, see also *Container.resources* attribute.

memory_mb

Requested memory amount in MegaBytes, `int`.

cpu

Requested number of CPUs, `float`. Please note, Docker supports fractions here, e.g. `0.5` CPU means a half or CPU on the target node.

gpu

The number of requested GPUs, `int`. Use `None` for jobs that doesn't require GPU.

gpu_model

The name of requested GPU model, `str` (or `None` for job without GPUs).

shm

Use Linux shared memory or not, `bool`. Provide `True` if you don't know what `/dev/shm` device means.

tpu_type

Requested TPU type, see also https://en.wikipedia.org/wiki/Tensor_processing_unit

tpu_software_version

Requested TPU software version.

StdStream

class neuro_sdk.**StdStream**

A class for communicating with attached job (*Jobs.attach()*) or exec session (*Jobs.exec()*). Use *read_out()* for reading from stdout/stderr and *write_in()* for writing into stdin.

coroutine close() → None

Close *StdStream* instance.

coroutine read_out() → Optional[*Message*]

Read next chunk from stdout/stderr.

Returns

Message instance for read data chunk or *None* if EOF is reached or *StdStream* was closed.

coroutine write_in(*data: bytes*) → None

Write *data* to stdin.

Parameters
data (*bytes*) – data to send.

Volume

class neuro_sdk.**Volume**

Read-only `dataclass` for describing mounted volumes of a container.

storage_uri

An URL on remotes storage, `yarl.URL`.

container_path

A path on container filesystem, `str`.

read_only

`True` is the volume is mounted in read-only mode, `False` for read-write (default).

SecretFile

class neuro_sdk.**SecretFile**

Read-only `dataclass` for describing secrets mounted as files in a container.

secret_uri

An URI on a secret, `yarl.URL`.

container_path

A path on container filesystem, `str`.

DiskVolume

class neuro_sdk.**DiskVolume**

Read-only `dataclass` for describing mounted disk volumes of a container.

disk_uri

An URI on a disk, `yarl.URL`.

container_path

A path on container filesystem, `str`.

read_only

`True` is the volume is mounted in read-only mode, `False` for read-write (default).

3.2.5 Storage API Reference

Storage

class neuro_sdk.**Storage**

Storage subsystem, available as *Client.storage*.

The subsystem can be used for listing remote storage, uploading and downloading files etc.

Remote filesystem operations

async-with async-for glob(*uri: URL*, *, *dironly: bool = False*) → AsyncContextManager[AsyncIterator[URL]]

Glob the given relative pattern *uri* in the directory represented by this *uri*, yielding all matching remote files (of any kind):

```
folder = yarl.URL("storage:folder/*")
async with client.storage.glob(folder) as uris:
    async for uri in uris:
        print(uri)
```

The **"**"** pattern means "this directory and all sub-directories, recursively". In other words, it enables recursive globbing:

```
folder = yarl.URL("storage:folder/**/*.py")
async with client.storage.glob(folder) as uris:
    async for uri in uris:
        print(uri)
```

Parameters

- **uri** (*URL*) – a pattern to glob.
- **fironly** (*bool*) – search in directory names only, `False` by default.

Returns

asynchronous iterator that can be used for enumerating found files and directories.

async-with async-for list(*uri: URL*) → AsyncContextManager[AsyncIterator[*FileStatus*]]

List a directory *uri* on the storage, e.g.:

```
folder = yarl.URL("storage:folder")
async with client.storage.list(folder) as statuses:
    async for status in statuses:
        print(status.name, status.size)
```

Parameters

uri (*URL*) – directory to list

Returns

asynchronous iterator which emits *FileStatus* objects for the directory content.

coroutine **mkdir**(*uri: URL*, *, *parents: bool = False*, *exist_ok: bool = False*) → None

Create a directory *uri*. Also create parent directories if *parents* is `True`, fail if directory exists and not *exist_ok*.

Parameters

- **uri** (*URL*) – a path to directory for creation, e.g. `yarl.URL("storage:folder/subfolder")`.
- **parents** (*bool*) – create parent directories if they are not exists, raise `FileNotFound` otherwise (`False` by default).
- **exist_ok** (*bool*) – finish successfully if directory *uri* already exists, raise `FileExistsError` otherwise (`False` by default).

Raises

`FileExistsError` if requested directory already exists and *exist_ok* flag is not set.

Raises

`FileNotFound` if parent directories don't exist and *parents* flag is not set.

coroutine **mv**(*src: URL*, *dst: URL*) → None

Rename remote file or directory *src* to *dst*.

Parameters

- **src** (*URL*) – remote source path, e.g. `yarl.URL("storage:folder/file.bin")`.
- **dst** (*URL*) – remote destination path, e.g. `yarl.URL("storage:folder/new.bin")`.

coroutine **rm**(*uri: URL*, *, *recursive: bool = False*) → None

Remove remote file or directory *uri*.

Parameters

- **uri** (*URL*) – remote path to delete, e.g. `yarl.URL("storage:folder/file.bin")`.
- **recursive** (*bool*) – remove a directory recursively with all nested files and folders if `True` (`False` by default).

Raises

`IsADirectoryError` if *uri* points on a directory and *recursive* flag is not set.

coroutine **stat**(*uri: URL*) → *FileStatus*

Return information about *uri*.

Parameters

uri (*URL*) – storage path for requested info, e.g. `yarl.URL("storage:folder/file.bin")`.

Returns

data structure for given *uri*, `FileStatus` object.

coroutine **disk_usage**(*cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → *DiskUsageInfo*

Return information about disk usage in given cluster and organization.

Parameters

- **cluster_name** (*str*) – cluster name to retrieve info. If `None` current cluster will be used.
- **org_name** (*str*) – Organization name to retrieve info. If `None` current organization will be used.

Returns
data structure for given cluster and organization, *DiskUsageInfo* object.

File operations

coroutine create(*uri: URL*, *data: AsyncIterator[bytes]*) → None

Create a file on storage under *uri* name, file it with a content from *data* asynchronous iterator, e.g.:

```
async def gen():
    for i in range(10):
        yield str(i) * 10

file = yarl.URL("storage:folder/file.bin")
source = gen()
await client.storage.create(file, source)
await source.aclose()
```

Parameters

- **uri** (*URL*) – path to created file, e.g. `yarl.URL("storage:folder/file.txt")`.
- **data** (*AsyncIterator[bytes]*) – asynchronous iterator used as data provider for file content.

coroutine write(*uri: URL*, *data: bytes*, *offset: int*) → None

Overwrite the part of existing file on storage under *uri* name.

Parameters

- **uri** (*URL*) – storage path of remote file, e.g. `yarl.URL("storage:folder/file.txt")`.
- **data** (*bytes*) – data to be written. Must be non-empty.
- **offset** (*int*) – position in file from which to write.

async-with async-for open(*uri: URL*, *offset: int = 0*, *size: Optional[int] = None*) → AsyncContextManager[AsyncIterator[bytes]]

Get the content of remote file *uri* or its fragment as asynchronous iterator, e.g.:

```
file = yarl.URL("storage:folder/file.txt")
async with client.storage.open(file) as content:
    async for data in content:
        print(data)
```

Parameters

- **uri** (*URL*) – storage path of remote file, e.g. `yarl.URL("storage:folder/file.txt")`.
- **offset** (*int*) – Position in file from which to read.
- **size** (*int*) – Maximal size of the read data. If `None` read to the end of the file.

Returns
asynchronous iterator used for retrieving the file content.

Copy operations

coroutine download_dir(*src: URL*, *dst: URL*, ***, *update: bool = False*, *continue_: bool = False*, *filter: Optional[Callable[[str], Awaitable[bool]]] = None*, *progress: Optional[AbstractRecursiveFileProgress] = None*) → None:

Recursively download remote directory *src* to local path *dst*.

Parameters

- **src** (*URL*) – path on remote storage to download a directory from e.g. `yarl.URL("storage:folder")`.
- **dst** (*URL*) – local path to save downloaded directory, e.g. `yarl.URL("file:///home/andrew/folder")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **continue** (*bool*) – if true, download only the part of the source file past the end of the destination file and append it to the destination file if the destination file is newer and not longer than the source file. Otherwise download and overwrite the whole file.
- **filter** (*Callable[[str], Awaitable[bool]]*) – a callback function for determining which files and subdirectories be downloaded. It is called with a relative path of file or directory and if the result is false the file or directory will be skipped.
- **progress** (`AbstractRecursiveFileProgress`) – a callback interface for reporting downloading progress, `None` for no progress report (default).

coroutine download_file(*src: URL*, *dst: URL*, ***, *update: bool = False*, *continue_: bool = False*, *progress: Optional[AbstractFileProgress] = None*) → None:

Download remote file *src* to local path *dst*.

Parameters

- **src** (*URL*) – path on remote storage to download a file from e.g. `yarl.URL("storage:folder/file.bin")`.
- **dst** (*URL*) – local path to save downloaded file, e.g. `yarl.URL("file:///home/andrew/folder/file.bin")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **continue** (*bool*) – if true, download only the part of the source file past the end of the destination file and append it to the destination file if the destination file is newer and not longer than the source file. Otherwise download and overwrite the whole file.
- **progress** (`AbstractFileProgress`) – a callback interface for reporting downloading progress, `None` for no progress report (default).

coroutine upload_dir(*src: URL*, *dst: URL*, ***, *update: bool = False*, *continue_: bool = False*, *filter: Optional[Callable[[str], Awaitable[bool]]] = None*, *ignore_file_names: AbstractSet[str] = frozenset()*, *progress: Optional[AbstractRecursiveFileProgress] = None*) → None:

Recursively upload local directory *src* to storage URL *dst*.

Parameters

- **src** (*URL*) – path to uploaded directory on local disk, e.g. `yarl.URL("file:///home/andrew/folder")`.

- **dst** (*URL*) – path on remote storage for saving uploading directory e.g. `yarl.URL("storage:folder")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **continue** (*bool*) – if true, upload only the part of the source file past the end of the destination file and append it to the destination file if the destination file is newer and not longer than the source file. Otherwise upload and overwrite the whole file.
- **filter** (*Callable[[str], Awaitable[bool]]*) – a callback function for determining which files and subdirectories be uploaded. It is called with a relative path of file or directory and if the result is false the file or directory will be skipped.
- **ignore_file_names** (*AbstractSet[str]*) – a set of names of files which specify filters for skipping files and subdirectories. The format of ignore files is the same as `.gitignore`.
- **progress** (`AbstractRecursiveFileProgress`) – a callback interface for reporting uploading progress, `None` for no progress report (default).

coroutine upload_file(*src: URL*, *dst: URL*, *, *update: bool = False*, *continue_: bool = False*, *progress: Optional[AbstractFileProgress] = None*) → None:

Upload local file *src* to storage URL *dst*.

Parameters

- **src** (*URL*) – path to uploaded file on local disk, e.g. `yarl.URL("file:///home/andrew/folder/file.txt")`.
- **dst** (*URL*) – path on remote storage for saving uploading file e.g. `yarl.URL("storage:folder/file.txt")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **continue** (*bool*) – if true, upload only the part of the source file past the end of the destination file and append it to the destination file if the destination file is newer and not longer than the source file. Otherwise upload and overwrite the whole file.
- **progress** (`AbstractFileProgress`) – a callback interface for reporting uploading progress, `None` for no progress report (default).

FileStatus

class neuro_sdk.**FileStatus**

Read-only `dataclass` for describing remote entry (file or directory).

modification_time

Modification time in seconds since the epoch, like the value returned from `time.time()`.

name

File or directory name, the last part of *path*.

permission

Permission (*read*, *write* or *manage*), *Action*.

path

Path to the entry, `str`.

size

File size in bytes, int.

type

Entry type, FileStatusType instance.

is_file() → bool

True if *type* is FileStatusType.FILE

is_dir() → bool

True if *type* is FileStatusType.DIRECTORY

AbstractFileProgress

class neuro_sdk.**AbstractFileProgress**

Base class for file upload/download progress, inherited from abc.ABC. User should inherit from this class and override abstract methods to get progress info from *Storage.upload_file()* and *Storage.download_file()* calls.

start(*data:* StorageProgressStart) → None

Called when transmission of the file starts.

Parameters
data (StorageProgressStart) – data for this event

step(*data:* StorageProgressStep) → None

Called when transmission of the file progresses (more bytes are transmitted).

Parameters
data (StorageProgressStep) – data for this event

complete(*data:* StorageProgressComplete) → None

Called when transmission of the file completes.

Parameters
data (StorageProgressComplete) – data for this event

AbstractRecursiveFileProgress

class neuro_sdk.**AbstractRecursiveFileProgress**

Base class for recursive file upload/download progress, inherited from *AbstractFileProgress*. User should inherit from this class and override abstract methods to get progress info from *Storage.upload_dir()* and *Storage.download_dir()* calls.

enter(*data:* StorageProgressEnterDir) → None

Called when recursive process enters directory.

Parameters
data (StorageProgressComplete) – data for this event

leave(*data:* StorageProgressLeaveDir) → None

Called when recursive process leaves directory. All files in that directory are now transmitted.

Parameters
data (StorageProgressLeaveDir) – data for this event

fail(*data:* StorageProgressFail) → None

Called when recursive process fails. It can happen because of touching special file (like block device file) or some other reason. Check **data.message** to get error details.

Parameters
data (StorageProgressFail) – data for this event

AbstractDeleteProgress

class neuro_sdk.**AbstractDeleteProgress**

Base class for file/directory delete progress, inherited from abc.ABC. User should inherit from this class and override abstract methods to get progress info from *Storage.rm()* calls.

delete(*data:* StorageProgressDelete) → None

Called when single item (either file or directory) was deleted. Directory removal happens after removal of all files and subdirectories is contains.

Parameters
data (StorageProgressDelete) – data for this event

StorageProgress event classes

class neuro_sdk.**StorageProgressStart**

src

yarl.URL of source file, e.g. URL("file:/home/user/upload.txt").

dst

yarl.URL of destination file, e.g. URL("storage://cluster/user/upload_to.txt").

size

Size of the transmitted file, in bytes, int.

class neuro_sdk.**StorageProgressStep**

src

yarl.URL of source file, e.g. URL("file:/home/user/upload.txt").

dst

yarl.URL of destination file, e.g. URL("storage://cluster/user/upload_to.txt").

current

Count of the transmitted bytes, int.

size

Size of the transmitted file, in bytes, int.

class neuro_sdk.**StorageProgressComplete**

src

yarl.URL of source file, e.g. URL("file:/home/user/upload.txt").

dst

yarl.URL of destination file, e.g. URL("storage://cluster/user/upload_to.txt").

size

Size of the transmitted file, in bytes, int.

class neuro_sdk.**StorageProgressEnterDir**

src

yarl.URL of source directory, e.g. URL("file:/home/user/upload/").

dst

yarl.URL of destination directory, e.g. URL("storage://cluster/user/upload_to/").

class neuro_sdk.**StorageProgressLeaveDir**

src

yarl.URL of source directory, e.g. URL("file:/home/user/upload/").

dst

yarl.URL of destination directory, e.g. URL("storage://cluster/user/upload_to/").

class neuro_sdk.**StorageProgressFail**

src

yarl.URL of source file that triggered error, e.g. URL("file:/dev/sda").

dst

yarl.URL of destination file, e.g. URL("storage://cluster/user/dev/sda.bin").

message

Explanation message for the error, str.

class neuro_sdk.**StorageProgressDelete**

uri

yarl.URL of the deleted file, e.g. URL("storage://cluster/user/delete_me.txt").

is_dir

True if removed item was a directory; **False** otherwise. bool

DiskUsageInfo

class neuro_sdk.**DiskUsageInfo**

Read-only dataclass for describing disk usage in particular cluster

cluster_name

Name of cluster, str.

total

Total storage size in bytes, int.

used

Used storage size in bytes, int.

free

Free storage size in bytes, int.

3.2.6 Images API Reference

Images

class neuro_sdk.**Images**

Docker image subsystem.

Used for pushing docker images onto Neuro docker registry for later usage by *Jobs.run()* and pulling these images back to local docker.

coroutine push(*local:* LocalImage, *remote: Optional[*RemoteImage*] = None*, **, progress: Optional[*AbstractDockerImageProgress*] = None*) → *RemoteImage*

Push *local* docker image to *remote* side.

Parameters

- **local** (LocalImage) – a spec of local docker image (e.g. created by `docker build`) for pushing on Neuro Registry.
- **remote** (RemoteImage) – a spec for remote image on Neuro Registry. Calculated from *local* image automatically if `None` (default).
- **progress** (AbstractDockerImageProgress) – a callback interface for reporting pushing progress, `None` for no progress report (default).

Returns

remote image if explicitly specified, calculated remote image if *remote* is `None` (*RemoteImage*)

coroutine pull(*remote: Optional[RemoteImage] = None*, *local: LocalImage*, **, progress: Optional[AbstractDockerImageProgress] = None*) → *RemoteImage*

Pull *remote* image from Neuro registry to *local* docker side.

Parameters

- **remote** (RemoteImage) – a spec for remote image on Neuro registry.
- **local** (LocalImage) – a spec of local docker image to pull. Calculated from *remote* image automatically if `None` (default).
- **progress** (AbstractDockerImageProgress) – a callback interface for reporting pulling progress, `None` for no progress report (default).

Returns

local image if explicitly specified, calculated remote image if *local* is `None` (*LocalImage*)

coroutine digest(*image:* RemoteImage) → str

Get digest of an image in Neuro registry.

Parameters

image (RemoteImage) – a spec for remote image on Neuro registry.

Returns

string representing image digest

coroutine rm(*image:* RemoteImage, *digest: str*) → str

Delete remote image specified by given reference and digest from Neuro registry.

Parameters

- **image** (RemoteImage) – a spec for remote image on Neuro registry.

- **digest** (*str*) – remote image digest, which can be obtained via *digest* method.

coroutine list(*cluster_name: Optional[str] = None*) → List[*RemoteImage*]

List images on Neuro registry available to the user.

Parameters
cluster_name (*str*) – name of the cluster.

None means the current cluster (default).

Returns
list of remote images not including tags (List[RemoteImage])

coroutine tags(*image:* RemoteImage) → List[*RemoteImage*]

List image references with tags for the specified remote *image*.

Parameters
image (RemoteImage) – a spec for remote image without tag on Neuro registry.

Returns
list of remote images with tags (List[RemoteImage])

coroutine size(*image:* RemoteImage) → int

Return image size.

Parameters
image (RemoteImage) – a spec for remote image with tag on Neuro registry.

Returns
remote image size in bytes

coroutine tag_info(*image:* RemoteImage) → *Tag*

Return info about specified tag.

Parameters
image (RemoteImage) – a spec for remote image with tag on Neuro registry.

Returns
tag information (name and size) (*Tag*)

AbstractDockerImageProgress

class neuro_sdk.**AbstractDockerImageProgress**

Base class for image operations progress, e.g. *Images.pull()* and *Images.push()*. Inherited from abc.ABC.

pull(*data:* ImageProgressPull) → None

Pulling image from remote Neuro registry to local Docker is started.

Parameters
data (ImageProgressPull) – additional data, e.g. local and remote image objects.

push(*data:* ImageProgressPush) → None

Pushing image from local Docker to remote Neuro registry is started.

Parameters
data (ImageProgressPush) – additional data, e.g. local and remote image objects.

step(*data:* ImageProgressStep) → None

Next step in image transfer is performed.

Parameters
data (ImageProgressStep) – additional data, e.g. image layer id and progress report.

ImageProgressPull

class neuro_sdk.**ImageProgressPull**

Read-only `dataclass` for pulling operation report.

src

Source image, *RemoteImage* instance.

dst

Destination image, *LocalImage* instance.

class neuro_sdk.**ImageProgressPush**

Read-only `dataclass` for pulling operation report.

src

Source image, *LocalImage* instance.

dst

Destination image, *RemoteImage* instance.

class neuro_sdk.**ImageProgressStep**

Read-only `dataclass` for push/pull progress step.

layer_id

Image layer, `str`.

message

Progress message, `str`.

LocalImage

class neuro_sdk.**LocalImage**

Read-only `dataclass` for describing *image* in local Docker system.

name

Image name, `str`.

tag

Image tag (`str`), `None` if the tag is omitted (implicit `latest` tag).

RemoteImage

class neuro_sdk.**RemoteImage**

Read-only `dataclass` for describing *image* in remote registry (Neuro Platform hosted or other registries like DockerHub).

name

Image name, `str`.

tag

Image tag (`str`), `None` if the tag is omitted (implicit `latest` tag).

owner

User name (`str`) of a person who manages this image.

Public DockerHub images (e.g. `"ubuntu:latest"`) have no *owner*, the attribute is `None`.

org_name

Name (`str`) of an organization who manages this image or *None* if there is no such org.

Public DockerHub images (e.g. `"ubuntu:latest"`) have no *org*, the attribute is `None`.

registry

Host name for images hosted on Neuro Registry (`str`), `None` for other registries like DockerHub.

as_docker_url(*with_scheme: bool = False*) → str

URL that can be used to reference this image with Docker.

Parameters

with_scheme (*bool*) – if set to True, returned URL includes scheme (*https://*), otherwise (default behavior) - scheme is omitted.

with_tag(*tag: bool*) → *RemoteImage*

Creates a new reference to remote image with *tag*.

Parameters

tag (*str*) – new tag

Returns

remote image with *tag*

classmethod new_neuro_image(*name: str*, *registry: str*, *, *owner: str*, *cluster_name: str*, *tag: Optional[str] = None*) → *RemoteImage*

Create a new instance referring to an image hosted on Neuro Platform.

Parameters

- **name** (*str*) – name of the image
- **registry** (*str*) – registry where the image is located
- **owner** (*str*) – image owner name
- **cluster_name** (*str*) – name of the cluster
- **tag** (*str*) – image tag

classmethod new_external_image(*name: str*, *registry: Optional[str] = None*, *, *tag: Optional[str] = None*) → *RemoteImage*

Create a new instance referring to an image hosted on an external registry (e.g. DockerHub).

Parameters

- **name** (*str*) – name of the image
- **registry** (*str*) – registry where the image is located
- **tag** (*str*) – image tag

class neuro_sdk.**Tag**

Read-only `dataclass` for tag information.

name

Tag name, `str`.

size

Tag size in bytes, `int`.

3.2.7 Users API Reference

Users

class neuro_sdk.**Users**

User management subsystem, available as *`Client.users`*.

coroutine **get_acl**(*user: str*, *scheme: Optional[str] = None*, *, *uri: Optional[URL] = None*) → Sequence[*Permission*]

Get a list of permissions for *user*.

Parameters

- **user** (*str*) – user name of person whom permissions are retrieved.
- **scheme** (*str*) – a filter to fetch permissions for specified URI scheme only, e.g. `"job"` or `"storage"`. Passing *scheme* is equivalent to passing `uri=scheme + ":"`.
- **uri** (*URL*) – a filter to fetch permissions for specified URI prefix only, e.g. `URL("job:")` or `URL("storage://mycluster/myname/mydir")`. You should specify full URI.

Returns

a `typing.Sequence` of *`Permission`* objects. Consider the return type as immutable list.

coroutine **get_shares**(*user: str*, *scheme: Optional[str] = None*, *, *uri: Optional[URL] = None*) → Sequence[*Share*]

Get resources shared with *user* by others.

Parameters

- **user** (*str*) – user name of person whom shares are retrieved.
- **scheme** (*str*) – a filter to fetch shares for specified URI scheme only, e.g. `"job"` or `"storage"`. Passing *scheme* is equivalent to passing `uri=scheme + ":"`.
- **uri** (*URL*) – a filter to fetch permissions for specified URI prefix only, e.g. `"job:"` or `"storage://mycluster/myname/mydir"`. You should specify full URI.

Returns

a `typing.Sequence` of *`Share`* objects. Consider the return type as immutable list.

coroutine **get_subroles**(*user: str*) → Sequence[str]

Get subroles of given *user*.

Parameters

user (*str*) – user name of person whom subroles are retrieved.

Returns

a `typing.Sequence` of `str` objects. Consider the return type as immutable list.

coroutine share(*user: str*, *permission:* Permission) → None

Share a resource specified by *permission* with *user*.

Parameters

- **user** (*str*) – user name to share a resource with.
- **permission** (Permission) – a new permission to add.

coroutine revoke(*user: str*, *uri: URL*) → None

Revoke all permissions for a resource specified by *uri* from *user*.

Parameters

- **user** (*str*) – user name to revoke a resource from.
- **uri** (*URL*) – a resource to revoke.

coroutine add(*role_name: str*) → None

Add new role.

Parameters

role_name (*str*) – role name. Components are separated by "/".

coroutine remove(*role_name: str*) → None

Remove existing role.

Parameters

role_name (*str*) – role name. Components are separated by "/".

Action

class neuro_sdk.**Action**

Enumeration that describes granted rights.

Can be one of the following values:

READ

Read-only access.

WRITE

Read and write access.

MANAGE

Full access: read, write and change access mode are allowed.

Share

class neuro_sdk.**Share**

Read-only `dataclass` for describing objects shared with user.

user

User name of person who shared a resource.

permission

Share specification (*uri* and *action*), *Permission*.

Permission

class neuro_sdk.**Permission**

Read-only `dataclass` for describing a resource.

uri

`yarl.URL` of resource, e.g. `URL("storage:folder")`

action

Access mode for resource, *Action* enumeration.

3.2.8 Secrets API Reference

Secrets

class neuro_sdk.**Secrets**

Secured secrets subsystems. Secrets can be passed as mounted files and environment variables into a running job.

async-with async-for list(*cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → AsyncContextManager[AsyncIterator[*Secret*]]

List user's secrets, async iterator. Yields *Secret* instances.

Parameters

- **cluster_name** (*str*) – cluster to list secrets. Default is current cluster.
- **org_name** (*str*) – org to list secrets. Default is current org.

coroutine add(*key: str*, *value: bytes*, *cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → None

Add a secret with name *key* and content *value*.

Parameters

- **key** (*str*) – secret's name.
- **vale** (*bytes*) – secret's value.
- **cluster_name** (*str*) – cluster to create a secret. Default is current cluster.
- **org_name** (*str*) – org to create a secrets. Default is current org.

coroutine rm(*key: str*, *cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → None

Delete a secret *key*.

Parameters

- **key** (*str*) – secret's name.
- **cluster_name** (*str*) – cluster to look for a secret. Default is current cluster.
- **org_name** (*str*) – org to look for a secrets. Default is current org.

Secret

class neuro_sdk.**Secret**

Read-only `dataclass` for describing secret instance.

key

The secret key, `str`.

owner

The secret owner username, `str`.

cluster_name

Cluster secret resource belongs to, `str`.

org_name

Org secret resource belongs to, `str` or *None* if there is no such org.

uri

URI of the secret resource, `yarl.URL`.

3.2.9 Disks API Reference

Disks

class neuro_sdk.**Disks**

Persistent disks subsystems. Disks can be passed as mounted volumes into a running job.

async-for list(*cluster_name: Optional[str] = None*) → AsyncContextManager[AsyncIterator[*Disk*]]

List user's disks, async iterator. Yields *Disk* instances.

Parameters

cluster_name (*str*) – cluster to list disks. Default is current cluster.

coroutine create(*storage: int*, *life_span: Optional[datetime.timedelta]*, *name: Optional[str]*, *cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → *Disk*

Create a disk with capacity of *storage* bytes.

Parameters

- **storage** (*int*) – storage capacity in bytes.
- **life_span** (*Optional[datetime.timedelta]*) – Duration of no usage after which disk will be deleted. `None` means no limit.
- **name** (*Optional[str]*) – Name of the disk. Should be unique among all user's disk.
- **cluster_name** (*str*) – cluster to create a disk. Default is current cluster.
- **org_name** (*str*) – org to create a disk. Default is current org.

Returns

Newly created disk info (*Disk*)

coroutine get(*disk_id_or_name: str*, *cluster_name: Optional[str] = None*) → *Disk*

Get a disk with id or name *disk_id_or_name*.

Parameters

- **disk_id_or_name** (*str*) – disk's id or name.

- **cluster_name** (*str*) – cluster to look for a disk. Default is current cluster.

Returns
Disk info (*Disk*)

coroutine rm(*disk_id_or_name: str*, *cluster_name: Optional[str] = None*) → None

Delete a disk with id or name *disk_id_or_name*.

Parameters

- **disk_id_or_name** (*str*) – disk's id or name.
- **cluster_name** (*str*) – cluster to look for a disk. Default is current cluster.

Disk

class neuro_sdk.**Disk**

Read-only `dataclass` for describing persistent disk instance.

id

The disk id, `str`.

storage

The disk capacity, in bytes, `int`.

used_bytes

The amount of used bytes on disk, `int` or `None` if this information is not available. Note that this field is updated periodically, so it can contain incorrect data.

owner

The disk owner username, `str`.

name

The disk name set by user, unique among all user's disks, `str` or `None` if no name was set.

status

Current disk status, *Disk.Status*.

uri

URI of the disk resource, `yarl.URL`.

cluster_name

Cluster disk resource belongs to, `str`.

org_name

Org disk resource belongs to, `str` or *None* if there is no such org.

created_at

Disk creation timestamp, `datetime`.

last_usage

Timestamp when disk was last attached to job, `datetime` or `None` if disk was never used.

timeout_unused

Max unused duration after which disk will be deleted by platform, `timedelta` or `None` if there is no limit.

Disk.Status

class Disk.**Status**

Enumeration that describes disk status.

Can be one of the following values:

PENDING

Disk is still creating. It can be attached to job, but job will not start until disk is created.

READY

Disk is created and ready to use.

BROKEN

Disk is broken and cannot be used anymore. Can happen if underneath storage device was destroyed.

3.2.10 ServiceAccounts API Reference

ServiceAccounts

class neuro_sdk.**ServiceAccounts**

Service accounts subsystems. Service accounts can be used to generate tokens that can be used in automated environments by third-party services.

async-with async-for list() → AsyncContextManager[AsyncIterator[*ServiceAccount*]]

List user's service accounts, async iterator. Yields `ServiceAccount` instances.

coroutine create(*name: Optional[str]*, *default_cluster: Optional[str]*) → Tuple[*ServiceAccount*, str]

Create a service account.

Parameters

- **role** (*str*) – Authorization role to use for this service account.
- **default_cluster** (*Optional[str]*) – Default cluster to embed into generated token. Defaults to current cluster.

Returns

Pair of newly created service account info and token. This is the only way to get token of a service account.

coroutine get(*id_or_name: str*) → *ServiceAccount*

Get a service account with id or name *id_or_name*.

Parameters

id_or_name (*str*) – service account's id or name.

Returns

Service account info (`ServiceAccount`)

coroutine rm(*id_or_name: str*) → None

Revoke and delete a service account with id or name *id_or_name*.

Parameters

id_or_name (*str*) – service account's id or name.

ServiceAccount

class neuro_sdk.**ServiceAccount**

Read-only `dataclass` for describing service account instance.

id

The service account id, `str`.

role

Authorization role this service account is based on.

owner

The service account owner username, `str`.

name

The service account name set by user, unique among all user's service accounts, `str` or `None` if no name was set.

default_cluster

A default cluster that this service account uses after login, `str`.

created_at

Service account creation timestamp, `datetime`.

role_deleted

`True` if corresponding role was deleted, otherwise `False`.

3.2.11 Buckets API Reference

Buckets

class neuro_sdk.**Buckets**

Blob storage buckets subsystems, available as `Client.buckets`.

The subsystem helps take advantage of many basic functionality of Blob Storage solutions different cloud providers support. For AWS it would be S3, for GCP - Cloud Storage, etc.

async-for list(*cluster_name: Optional[str] = None*) → AsyncContextManager[AsyncIterator[*Bucket*]]

List user's buckets, async iterator. Yields `Bucket` instances.

Parameters
cluster_name (*str*) – cluster to list buckets. Default is current cluster.

coroutine create(*name: Optional[str]*, *cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → *Bucket*

Create a new bucket.

Parameters

- **name** (*Optional[str]*) – Name of the bucket. Should be unique among all user's bucket.
- **cluster_name** (*str*) – cluster to create a bucket. Default is current cluster.
- **org_name** (*str*) – org to create a bucket. Default is current org.

Returns
Newly created bucket info (*Bucket*)

coroutine import_external(*provider: Bucket.Provider*, *provider_bucket_name: str*, *credentials: Mapping[str, str]*, *name: Optional[str] = None*, *cluster_name: Optional[str] = None*, *org_name: Optional[str] = None*) → *Bucket*

Import a new bucket.

Parameters

- **provider** (*Bucket.Provider*) – Provider type of imported bucket.
- **provider_bucket_name** (*str*) – Name of external bucket inside the provider.
- **credentials** (*Mapping[str, str]*) – Raw credentials to access bucket provider.
- **name** (*Optional[str]*) – Name of the bucket. Should be unique among all user's bucket.
- **cluster_name** (*str*) – cluster to import a bucket. Default is current cluster.
- **org_name** (*str*) – org to import a bucket. Default is current org.

Returns

Newly imported bucket info (*Bucket*)

coroutine get(*bucket_id_or_name: str*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str] = None*) → *Bucket*

Get a bucket with id or name *bucket_id_or_name*.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

Returns

Bucket info (*Bucket*)

coroutine rm(*bucket_id_or_name: str*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str] = None*) → None

Delete a bucket with id or name *bucket_id_or_name*.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

coroutine request_tmp_credentials(*bucket_id_or_name: str*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str] = None*) → *BucketCredentials*

Get a temporary provider credentials to bucket with id or name *bucket_id_or_name*.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

Returns
Bucket credentials info (*BucketCredentials*)

coroutine set_public_access(*bucket_id_or_name: str*, *public_access: bool*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str) = None*) → *Bucket*

Enable or disable public (anonymous) read access to bucket.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **public_access** (*str*) – New public access setting.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

Returns
Bucket info (*Bucket*)

coroutine head_blob(*bucket_id_or_name: str*, *key: str*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str) = None*) → *BucketEntry*

Look up the blob and return it's metadata.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **key** (*str*) – key of the blob.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

Returns
BucketEntry object.

Raises
ResourceNotFound if key does not exist.

coroutine put_blob(*bucket_id_or_name: str, key: str, body: Union[AsyncIterator[bytes], bytes], cluster_name: Optional[str] = None, bucket_owner: Optional[str) = None,)* → None

Create or replace blob identified by key in the bucket, e.g:

```
large_file = Path("large_file.dat")
size = large_file.stat().st_size
file_md5 = await calc_md5(large_file)

async def body_stream():
    with large_file.open("r") as f:
        for line in f:
            yield f

await client.buckets.put_blob(
    bucket_id_or_name="my_bucket",
    key="large_file.dat",
    body=body_stream,
)
```

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **key** (*str*) – Key of the blob.
- **body** (*bytes*) – Body of the blob. Can be passed as either `bytes` or as an `AsyncIterator[bytes]`.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

coroutine fetch_blob(*bucket_id_or_name: str*, *key: str*, *offset: int = 0*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str] = None*) → AsyncIterator[bytes]

Look up the blob and return it's body content only. The content will be streamed using an asynchronous iterator, e.g.:

```
async with client.buckets.fetch_blob("my_bucket", key="file.txt") as content:
    async for data in content:
        print("Next chunk of data:", data)
```

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **key** (*str*) – Key of the blob.
- **offset** (*int*) – Position in blob from which to read.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

coroutine delete_blob(*bucket_id_or_name: str*, *key: str*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str] = None*) → None

Remove blob from the bucket.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **key** (*str*) – key of the blob.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

coroutine list_blobs(*uri: URL*, *recursive: bool = False*, *limit: int = 10000*) → AsyncContextManager[AsyncIterator[*BucketEntry*]]

List blobs in the bucket. You can filter by prefix and return results similar to a folder structure if `recursive=False` is provided.

Parameters

- **uri** (*URL*) – URL that specifies bucket and prefix to list blobs, e.g. `yarl.URL("blob:bucket_name/path/in/bucket")`.

- **bool** (*recursive*) – If `True` listing will contain *all* keys filtered by prefix, while with `False` only ones up to next `/` will be returned. To indicate missing keys, all that were listed will be combined under a common prefix and returned as *BlobCommonPrefix*.
- **int** (*limit*) – Maximum number of *BucketEntry* objects returned.

coroutine glob_blobs(*uri: URL*) → AsyncContextManager[AsyncIterator[*BucketEntry*]]

Glob search for blobs in the bucket:

```
async with client.buckets.glob_blobs(
    uri=URL("blob:my_bucket/folder1/**/*.txt")
) as blobs:
    async for blob in blobs:
        print(blob.key)
```

Similar to *Storage.glob()* the "`**`" pattern means "this directory and all sub-directories, recursively".

Parameters

uri (*URL*) – URL that specifies bucket and pattern to glob blobs, e.g. `yarl.URL("blob:bucket_name/path/**/*.bin")`.

coroutine upload_file(*src: URL*, *dst: URL*, *, *update: bool = False*, *progress: Optional[AbstractFileProgress] = None*) → None:

Similarly to *Storage.upload_file()*, allows to upload local file *src* to bucket URL *dst*.

Parameters

- **src** (*URL*) – path to uploaded file on local disk, e.g. `yarl.URL("file:///home/andrew/folder/file.txt")`.
- **dst** (*URL*) – URL that specifies bucket and key to upload file e.g. `yarl.URL("blob:bucket_name/folder/file.txt")`.
- **update** (*bool*) – if true, upload only when the source file is newer than the destination file or when the destination file is missing.
- **progress** (`AbstractFileProgress`) – a callback interface for reporting uploading progress, `None` for no progress report (default).

coroutine download_file(*src: URL*, *dst: URL*, *, *update: bool = False*, *continue_: bool = False*, *progress: Optional[AbstractFileProgress] = None*) → None:

Similarly to *Storage.download_file()*, allows to download remote file *src* to local path *dst*.

Parameters

- **src** (*URL*) – URL that specifies bucket and blob key to download e.g. `yarl.URL("blob:bucket_name/folder/file.bin")`.
- **dst** (*URL*) – local path to save downloaded file, e.g. `yarl.URL("file:///home/andrew/folder/file.bin")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **continue** (*bool*) – if true, download only the part of the source file past the end of the destination file and append it to the destination file if the destination file is newer and not longer than the source file. Otherwise download and overwrite the whole file.
- **progress** (`AbstractFileProgress`) – a callback interface for reporting downloading progress, `None` for no progress report (default).

coroutine upload_dir(*src: URL*, *dst: URL*, ***, *update: bool = False*, *filter: Optional[Callable[[str], Awaitable[bool]]] = None*, *ignore_file_names: AbstractSet[str] = frozenset()*, *progress: Optional[AbstractRecursiveFileProgress] = None*) → None:

Similarly to *Storage.upload_dir()*, allows to recursively upload local directory *src* to Blob Storage URL *dst*.

Parameters

- **src** (*URL*) – path to uploaded directory on local disk, e.g. `yarl.URL("file:///home/andrew/folder")`.
- **dst** (*URL*) – path on Blob Storage for saving uploading directory e.g. `yarl.URL("blob:bucket_name/folder/")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **filter** (*Callable[[str], Awaitable[bool]]*) – a callback function for determining which files and subdirectories be uploaded. It is called with a relative path of file or directory and if the result is false the file or directory will be skipped.
- **ignore_file_names** (*AbstractSet[str]*) – a set of names of files which specify filters for skipping files and subdirectories. The format of ignore files is the same as `.gitignore`.
- **progress** (AbstractRecursiveFileProgress) – a callback interface for reporting uploading progress, `None` for no progress report (default).

coroutine download_dir(*src: URL*, *dst: URL*, ***, *update: bool = False*, *continue_: bool = False*, *filter: Optional[Callable[[str], Awaitable[bool]]] = None*, *progress: Optional[AbstractRecursiveFileProgress] = None*) → None:

Similarly to *Storage.download_dir()*, allows to recursively download remote directory *src* to local path *dst*.

Parameters

- **src** (*URL*) – path on Blob Storage to download a directory from e.g. `yarl.URL("blob:bucket_name/folder/")`.
- **dst** (*URL*) – local path to save downloaded directory, e.g. `yarl.URL("file:///home/andrew/folder")`.
- **update** (*bool*) – if true, download only when the source file is newer than the destination file or when the destination file is missing.
- **continue** (*bool*) – if true, download only the part of the source file past the end of the destination file and append it to the destination file if the destination file is newer and not longer than the source file. Otherwise download and overwrite the whole file.
- **filter** (*Callable[[str], Awaitable[bool]]*) – a callback function for determining which files and subdirectories be downloaded. It is called with a relative path of file or directory and if the result is false the file or directory will be skipped.
- **progress** (AbstractRecursiveFileProgress) – a callback interface for reporting downloading progress, `None` for no progress report (default).

coroutine blob_is_dir(*uri: URL*) → bool

Check weather *uri* specifies a "folder" blob in a bucket.

Parameters

src (*URL*) – URL that specifies bucket and blob key e.g. `yarl.URL("blob:bucket_name/folder/sub_folder")`.

coroutine blob_rm(*uri: URL*, *, *recursive: bool = False*, *progress: Optional[AbstractDeleteProgress] = None*) → None

Remove blobs from bucket.

Parameters

- **uri** (*URL*) – URL that specifies bucket and blob key e.g. `yarl.URL("blob:bucket_name/folder/sub_folder")`.
- **recursive** (*bool*) – remove a directory recursively with all nested files and folders if `True` (`False` by default).
- **progress** (`AbstractDeleteProgress`) – a callback interface for reporting delete progress, `None` for no progress report (default).

Raises

`IsADirectoryError` if *uri* points on a directory and *recursive* flag is not set.

coroutine make_signed_url(*uri: URL*, *expires_in_seconds: int = 3600*) → URL

Generate a singed url that allows temporary access to blob.

Parameters

- **uri** (*URL*) – URL that specifies bucket and blob key e.g. `yarl.URL("blob:bucket_name/folder/file.bin")`.
- **expires_in_seconds** (*int*) – Duration in seconds generated url will be valid.

Returns

Signed url (`yarl.URL`)

coroutine get_disk_usage(*bucket_id_or_name: str*, *cluster_name: Optional[str] = None*, *bucket_owner: Optional[str) = None*) → AsyncContextManager[AsyncIterator[*BucketUsage*]]

Get disk space usage of a given bucket. Iterator yield partial results as calculation for the whole bucket can take time.

Parameters

- **bucket_id_or_name** (*str*) – bucket's id or name.
- **cluster_name** (*str*) – cluster to look for a bucket. Default is current cluster.
- **bucket_owner** (*str*) – bucket owner's username. Used only if looking up for bucket by it's name. Default is current user.

async-for persistent_credentials_list(*cluster_name: Optional[str] = None*) → AsyncContextManager[AsyncIterator[*PersistentBucketCredentials*]]

List user's bucket persistent credentials, async iterator. Yields `PersistentBucketCredentials` instances.

Parameters

cluster_name (*str*) – cluster to list persistent credentials. Default is current cluster.

coroutine persistent_credentials_create(*bucket_ids: Iterable[str]*, *name: Optional[str]*, *read_only: Optional[bool] = False*, *cluster_name: Optional[str] = None*) → *PersistentBucketCredentials*

Create a new persistent credentials for given set of buckets.

Parameters

- **bucket_ids** (*Iterable[str]*) – Iterable of bucket ids to create credentials for.

- **name** (*Optional[str]*) – Name of the persistent credentials. Should be unique among all user's bucket persistent credentials.
- **read_only** (*str*) – Allow only read-only access using created credentials. `False` by default.
- **cluster_name** (*str*) – cluster to create a persistent credentials. Default is current cluster.

Returns
Newly created credentials info (*PersistentBucketCredentials*)

coroutine persistent_credentials_get(*credential_id_or_name: str*, *cluster_name: Optional[str] = None*) → *PersistentBucketCredentials*

Get a persistent credentials with id or name *credential_id_or_name*.

Parameters

- **credential_id_or_name** (*str*) – persistent credentials's id or name.
- **cluster_name** (*str*) – cluster to look for a persistent credentials. Default is current cluster.

Returns
Credentials info (*PersistentBucketCredentials*)

coroutine persistent_credentials_rm(*credential_id_or_name: str*, *cluster_name: Optional[str] = None*) → None

Delete a persistent credentials with id or name *credential_id_or_name*.

Parameters

- **credential_id_or_name** (*str*) – persistent credentials's id or name.
- **cluster_name** (*str*) – cluster to look for a persistent credentials. Default is current cluster.

Bucket

class neuro_sdk.**Bucket**

Read-only `dataclass` for describing single bucket.

id
The bucket id, `str`.

owner
The bucket owner username, `str`.

name
The bucket name set by user, unique among all user's buckets, `str` or `None` if no name was set.

uri
URI of the bucket resource, `yarl.URL`.

cluster_name
Cluster this bucket belongs to, `str`.

org_name
Org this bucket belongs to, `str` or *None* if there is no such org.

created_at

Bucket creation timestamp, datetime.

provider

Blob storage provider this bucket belongs to, *Bucket.Provider*.

BucketCredentials

class neuro_sdk.**BucketCredentials**

Read-only dataclass for describing credentials to single bucket.

bucket_id

The bucket id, str.

provider

Blob storage provider this bucket belongs to, *Bucket.Provider*.

credentials

Raw credentials to access a bucket inside the provider, Mapping[str, str]

Bucket.Provider

class Bucket.**Provider**

Enumeration that describes bucket providers.

Can be one of the following values:

AWS

Amazon Web Services S3 bucket

MINIO

Minio S3 bucket

AZURE

Azure blob storage container

PersistentBucketCredentials

class neuro_sdk.**PersistentBucketCredentials**

Read-only dataclass for describing persistent credentials to some set of buckets created after user request.

id

The credentials id, str.

owner

The credentials owner username, str.

name

The credentials name set by user, unique among all user's bucket credentials, str or None if no name was set.

read_only

The credentials provide read-only access to buckets, bool.

cluster_name

Cluster this credentials belongs to, str.

credentials

List of per bucket credentials, List[BucketCredentials]

BucketEntry

class neuro_sdk.**BucketEntry**

An abstract class dataclass for describing bucket contents entries.

key

Key of the blob, str.

bucket

Containing bucket, *Bucket*.

size

Size of the data in *bytes*, int.

created_at

Blob creation timestamp, datetime or None if underlying blob engine do not store such information

modified_at

Blob modification timestamp, datetime or None if underlying blob engine do not store such information

uri

URI identifying the blob, URL, e.g. blob://cluster_name/username/my_bucket/file.txt.

name

Name of blob, part of key after last /, str

is_dir(*uri: URL*) → bool

True if entry is directory blob object

is_file(*uri: URL*) → bool

True if entry is file blob object

BlobObject

class neuro_sdk.**BlobObject**

An ancestor of *BucketEntry* used for key that are present directly in underlying blob storage.

BlobCommonPrefix

class neuro_sdk.**BlobCommonPrefix**

An ancestor of *BucketEntry* for describing common prefixes for blobs in non-recursive listing. You can treat it as a kind of *folder* on Blob Storage.

BucketUsage

class neuro_sdk.**BucketUsage**

An `dataclass` for describing bucket disk space usage.

total_bytes

Total size of all objects in bytes, `int`.

object_count

Total number of objects, `int`.

3.2.12 Parser Reference

Parser

class neuro_sdk.**Parser**

A set of parsing helpers, useful for building helper dataclasses from string representations.

volume(*volume: str*) → *Volume*

Parse *volume* string into `Volume` object.

The string is a three fields separated by colon characters (`:`): `<storage-uri:container-path[:ro]>`.

storage-uri is a URL on local storage, e.g. `storage:folder` points on `<user-root>/folder` directory.

container-path is a path inside a job where *storage-url* is mounted.

Optional *ro* means that *storage-url* is mounted in *read-only* mode. Writable mode is used if *ro* is omitted.

Raise
`ValueError` if *volume* has invalid format.

local_image(*image: str*) → *LocalImage*

Parse *image* string into `LocalImage`.

The string should fit to the following pattern: `name[:tag]`, e.g. `"ubuntu:latest"`.

Raise
`ValueError` if *image* has invalid format.

remote_image(*image: str*) → *RemoteImage*

Parse *image* string into `RemoteImage`.

The string should fit to `name[:tag]` or `image:name[tag]` patterns, e.g. `"ubuntu:latest"` or `image:my-image:latest`. The former is used for public DockerHub images, the later is for Neuro image registry.

Raise
`ValueError` if *image* has invalid format.

envs(*env: Sequence[str]*, *env_file: Sequence[str] = ()*) → *EnvParseResult*

Parse a sequence of *env* variables and a sequence of *env_file* file names.

Parameters

- **env** (`Sequence[str]`) – Sequence of *env* variable specification. Each element can be either: - *ENV_NAME*. Current system *env* variable value will be used. Defaults to empty string. - *ENV_NAME=VALUE*. Given value will be used.

- **env_file** (*Sequence[str]*) – Sequence of `.env` files to use. File content processed same way as *env* parameter.

Returns

EnvParseResult with parsing result

volumes(*volume: Sequence[str]*) → *VolumeParseResult*

Parse a sequence of volume definition into a tuple of three mappings - first one for all regular volumes, second one for volumes using secrets and third for disk volumes.

Parameters

env (*Sequence[str]*) – Sequence of volumes specification. Each element can be either: - *STORAGE_URI:MOUNT_PATH:RW_FLAG*. - *SECRET_URI:MOUNT_PATH*. - *DISK_URI:MOUNT_PATH:RW_FLAG*.

Returns

VolumeParseResult with parsing result

uri_to_str(*uri: URL*) → str

Convert `URL` object into `str`.

str_to_uri(*uri: str*, *, *allowed_schemes: Iterable[str] = ()*, *cluster_name: Optional[str] = None*, *short: bool = False*) → URL

Parse a string into *normalized* `URL` for future usage by SDK methods.

Parameters

- **uri** (*str*) – an URI (`'storage:folder/file.txt'`) or local file path (`'/home/user/folder/file.txt'`) to parse.
- **allowed_schemes** (*Iterable[str]*) – an *iterable* of accepted URI schemes, e.g. `('file', 'storage')`. No scheme check is performed by default.
- **cluster_name** (*Optional[str]*) – optional cluster name, the default cluster is used if not specified.
- **short** (*bool*) – if `True`, return short URL (without cluster and user names if possible). `False` by default.

Returns

`URL` with parsed URI.

Raises

ValueError – if `uri` is invalid or provides a scheme not enumerated by `allowed_schemes` argument.

uri_to_path(*uri: URL*, *, *cluster_name: Optional[str] = None*) → Path

Convert `URL` into `Path`.

Raises

ValueError – if `uri` has no `'file:'` scheme.

path_to_uri(*path: Path*) → URL

Convert `Path` object into *normalized* `URL` with `'file:'` scheme.

Parameters

path (*Path*) – a path to convert.

Returns

`URL` that represent a `path`.

normalize_uri(*uri: URL*, *, *allowed_schemes: Iterable[str] = ()*, *cluster_name: Optional[str] = None*, *short: bool = False*) → URL

Normalize `uri` according to current user name, cluster and allowed schemes.

Normalized form has two variants:

1. Long form: cluster and user names are always present, e.g. *storage://cluster/user/dir/file.txt*.
2. Short form: cluster and user are omitted if they are equal to default values given from *client.config.cluster_name* and *client.config.username*, e.g. *storage:dir/file.txt*.

Parameters
- **uri** (*URL*) – an URI to normalize.
- **allowed_schemes** (*Iterable[str]*) – an *iterable* of accepted URI schemes, e.g. `('file', 'storage')`. No scheme check is performed by default.
- **cluster_name** (*Optional[str]*) – optional cluster name, the default cluster is used if not specified.
- **short** (*bool*) – if `True`, return short URL (without cluster and user names if possible). `False` by default.

Returns
`URL` with normalized URI.

Raises
ValueError – if `uri` is invalid or provides a scheme not enumerated by `allowed_schemes` argument.

EnvParseResult

class neuro_sdk.**EnvParseResult**

env

Mapping of parsed environmental variables, `Dict[str, str]`.

secret_env

Mapping of parsed using secrets environmental variables, `Dict[str, URL]`.

VolumeParseResult

class neuro_sdk.**VolumeParseResult**

volumes

List of parsed regular volumes, `Sequence[str]`.

secret_files

List of parsed secret files, `List[SecretFile]`.

disk_volumes

List of parsed disk volumes, `List[DiskVolume]`.

3.2.13 Plugins API Reference

PluginManager

class neuro_sdk.**PluginManager**

Allows plugins to register their features. Provided to **neuro_api** entrypoint (check https://packaging.python.org/specifications/entry-points/ for more info about entry points).

config

Define new user config parameters *ConfigBuilder*.

ConfigBuilder

class neuro_sdk.**ConfigBuilder**

Helper class that contains methods to define new user config variables (check *Config.get_user_config()*).

define_int(*section: str*, *name: str*) → None

Define new integer config parameter with given name in given section

define_bool(*section: str*, *name: str*) → None

Define new *bool* config parameter with given name in given section

define_float(*section: str*, *name: str*) → None

Define new float config parameter with given name in given section

define_str(*section: str*, *name: str*) → None

Define new string config parameter with given name in given section

define_int_list(*section: str*, *name: str*) → None

Define new integer list config parameter with given name in given section

define_bool_list(*section: str*, *name: str*) → None

Define new *bool* list config parameter with given name in given section

define_str_list(*section: str*, *name: str*) → None

Define new string list config parameter with given name in given section

define_float_list(*section: str*, *name: str*) → None

Define new float list config parameter with given name in given section

3.2.14 Glossary

CLI

Command line interface, a text interface program executed from a terminal (*bash*, *cmd.exe*, *PowerShell*, *zsh* etc.)

tty

A mode when local terminal is connected to remote shell executed in a job. Any key pressed by user is sent to remote shell, any print to a screen from remote shell is displayed on local terminal.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n
neuro_sdk, 11

INDEX

Symbols

_internal (*neuro_sdk.JobDescription attribute*), 26

A

AbstractDeleteProgress (*class in neuro_sdk*), 38
AbstractDockerImageProgress (*class in neuro_sdk*), 41
AbstractFileProgress (*class in neuro_sdk*), 37
AbstractRecursiveFileProgress (*class in neuro_sdk*), 37
Action (*class in neuro_sdk*), 45
action (*neuro_sdk.Permission attribute*), 46
active_items() (*neuro_sdk.JobStatus method*), 27
add() (*neuro_sdk.Secrets method*), 46
add() (*neuro_sdk.Users method*), 45
ALWAYS (*neuro_sdk.JobRestartPolicy attribute*), 26
api_url (*neuro_sdk.Config attribute*), 16
as_docker_url() (*neuro_sdk.RemoteImage method*), 43
attach() (*neuro_sdk.Jobs method*), 18
AWS (*neuro_sdk.Bucket.Provider attribute*), 58
AZURE (*neuro_sdk.Bucket.Provider attribute*), 58

B

blob_is_dir() (*neuro_sdk.Buckets method*), 55
blob_rm() (*neuro_sdk.Buckets method*), 55
BlobCommonPrefix (*class in neuro_sdk*), 59
BlobObject (*class in neuro_sdk*), 59
BROKEN (*neuro_sdk.Disk.Status attribute*), 49
Bucket (*class in neuro_sdk*), 57
bucket (*neuro_sdk.BucketEntry attribute*), 59
Bucket.Provider (*class in neuro_sdk*), 58
bucket_id (*neuro_sdk.BucketCredentials attribute*), 58
BucketCredentials (*class in neuro_sdk*), 58
BucketEntry (*class in neuro_sdk*), 59
Buckets (*class in neuro_sdk*), 50
BucketUsage (*class in neuro_sdk*), 60
bump_life_span() (*neuro_sdk.Jobs method*), 24

C

CANCELLED (*neuro_sdk.JobStatus attribute*), 27
CLI, **63**
Client (*class in neuro_sdk*), 14
close() (*neuro_sdk.Client method*), 14
close() (*neuro_sdk.StdStream method*), 30
closed (*neuro_sdk.Client attribute*), 15
Cluster (*class in neuro_sdk*), 17
cluster_name (*neuro_sdk.Bucket attribute*), 57
cluster_name (*neuro_sdk.Config attribute*), 15
cluster_name (*neuro_sdk.Disk attribute*), 48
cluster_name (*neuro_sdk.DiskUsageInfo attribute*), 39
cluster_name (*neuro_sdk.JobDescription attribute*), 25
cluster_name (*neuro_sdk.PersistentBucketCredentials attribute*), 58
cluster_name (*neuro_sdk.Secret attribute*), 47
clusters (*neuro_sdk.Config attribute*), 15
command (*neuro_sdk.Container attribute*), 24
complete() (*neuro_sdk.AbstractFileProgress method*), 37
Config (*class in neuro_sdk*), 15
config (*neuro_sdk.Client attribute*), 14
config (*neuro_sdk.PluginManager attribute*), 62
ConfigBuilder (*class in neuro_sdk*), 63
Container (*class in neuro_sdk*), 24
container (*neuro_sdk.JobDescription attribute*), 25
container_path (*neuro_sdk.DiskVolume attribute*), 31
container_path (*neuro_sdk.SecretFile attribute*), 31
container_path (*neuro_sdk.Volume attribute*), 31
cpu (*neuro_sdk.JobTelemetry attribute*), 29
cpu (*neuro_sdk.Preset attribute*), 17
cpu (*neuro_sdk.Resources attribute*), 30
create() (*neuro_sdk.Buckets method*), 50
create() (*neuro_sdk.Disks method*), 47
create() (*neuro_sdk.ServiceAccounts method*), 49
create() (*neuro_sdk.Storage method*), 34
created_at (*neuro_sdk.Bucket attribute*), 57
created_at (*neuro_sdk.BucketEntry attribute*), 59
created_at (*neuro_sdk.Disk attribute*), 48
created_at (*neuro_sdk.JobStatusHistory attribute*), 29
created_at (*neuro_sdk.ServiceAccount attribute*), 50
credentials (*neuro_sdk.BucketCredentials attribute*), 58
credentials (*neuro_sdk.PersistentBucketCredentials attribute*), 59

current (*neuro_sdk.StorageProgressStep attribute*), 38

D

data (*neuro_sdk.Message attribute*), 30
default_cluster (*neuro_sdk.ServiceAccount attribute*), 50
define_bool() (*neuro_sdk.ConfigBuilder method*), 63
define_bool_list() (*neuro_sdk.ConfigBuilder method*), 63
define_float() (*neuro_sdk.ConfigBuilder method*), 63
define_float_list() (*neuro_sdk.ConfigBuilder method*), 63
define_int() (*neuro_sdk.ConfigBuilder method*), 63
define_int_list() (*neuro_sdk.ConfigBuilder method*), 63
define_str() (*neuro_sdk.ConfigBuilder method*), 63
define_str_list() (*neuro_sdk.ConfigBuilder method*), 63
delete() (*neuro_sdk.AbstractDeleteProgress method*), 38
delete_blob() (*neuro_sdk.Buckets method*), 53
description (*neuro_sdk.JobDescription attribute*), 25
description (*neuro_sdk.JobStatusHistory attribute*), 28
description (*neuro_sdk.JobStatusItem attribute*), 28
digest() (*neuro_sdk.Images method*), 40
Disk (*class in neuro_sdk*), 48
Disk.Status (*class in neuro_sdk*), 49
disk_uri (*neuro_sdk.DiskVolume attribute*), 31
disk_usage() (*neuro_sdk.Storage method*), 33
disk_volumes (*neuro_sdk.Container attribute*), 24
disk_volumes (*neuro_sdk.VolumeParseResult attribute*), 62
Disks (*class in neuro_sdk*), 47
disks (*neuro_sdk.Client attribute*), 14
DiskUsageInfo (*class in neuro_sdk*), 39
DiskVolume (*class in neuro_sdk*), 31
download_dir() (*neuro_sdk.Buckets method*), 55
download_dir() (*neuro_sdk.Storage method*), 35
download_file() (*neuro_sdk.Buckets method*), 54
download_file() (*neuro_sdk.Storage method*), 35
dst (*neuro_sdk.ImageProgressPull attribute*), 42
dst (*neuro_sdk.ImageProgressPush attribute*), 42
dst (*neuro_sdk.StorageProgressComplete attribute*), 38
dst (*neuro_sdk.StorageProgressEnterDir attribute*), 39
dst (*neuro_sdk.StorageProgressFail attribute*), 39
dst (*neuro_sdk.StorageProgressLeaveDir attribute*), 39
dst (*neuro_sdk.StorageProgressStart attribute*), 38
dst (*neuro_sdk.StorageProgressStep attribute*), 38

E

enter() (*neuro_sdk.AbstractRecursiveFileProgress method*), 37
entrypoint (*neuro_sdk.Container attribute*), 24
env (*neuro_sdk.Container attribute*), 24
env (*neuro_sdk.EnvParseResult attribute*), 62
EnvParseResult (*class in neuro_sdk*), 62
envs() (*neuro_sdk.Parser method*), 60
exec() (*neuro_sdk.Jobs method*), 18
exit_code (*neuro_sdk.JobStatusHistory attribute*), 29
exit_code (*neuro_sdk.JobStatusItem attribute*), 28

F

Factory (*class in neuro_sdk*), 12
fail() (*neuro_sdk.AbstractRecursiveFileProgress method*), 37
FAILED (*neuro_sdk.JobStatus attribute*), 27
fetch() (*neuro_sdk.Config method*), 15
fetch_blob() (*neuro_sdk.Buckets method*), 53
fileno (*neuro_sdk.Message attribute*), 30
FileStatus (*class in neuro_sdk*), 36
find_project_root() (*in module neuro_sdk*), 15
finished_at (*neuro_sdk.JobStatusHistory attribute*), 29
finished_items() (*neuro_sdk.JobStatus method*), 27
free (*neuro_sdk.DiskUsageInfo attribute*), 39

G

get() (*in module neuro_sdk*), 11
get() (*neuro_sdk.Buckets method*), 51
get() (*neuro_sdk.Disks method*), 47
get() (*neuro_sdk.Factory method*), 12
get() (*neuro_sdk.ServiceAccounts method*), 49
get_acl() (*neuro_sdk.Users method*), 44
get_capacity() (*neuro_sdk.Jobs method*), 19
get_disk_usage() (*neuro_sdk.Buckets method*), 56
get_shares() (*neuro_sdk.Users method*), 44
get_subroles() (*neuro_sdk.Users method*), 44
get_user_config() (*neuro_sdk.Config method*), 16
glob() (*neuro_sdk.Storage method*), 32
glob_blobs() (*neuro_sdk.Buckets method*), 54
gpu (*neuro_sdk.Preset attribute*), 17
gpu (*neuro_sdk.Resources attribute*), 30
gpu_duty_cycle (*neuro_sdk.JobTelemetry attribute*), 29
gpu_memory (*neuro_sdk.JobTelemetry attribute*), 29
gpu_model (*neuro_sdk.Preset attribute*), 18
gpu_model (*neuro_sdk.Resources attribute*), 30

H

head_blob() (*neuro_sdk.Buckets method*), 52
HIGH (*neuro_sdk.JobPriority attribute*), 27
history (*neuro_sdk.JobDescription attribute*), 25
http (*neuro_sdk.Container attribute*), 24
http_url (*neuro_sdk.JobDescription attribute*), 26
HTTPPort (*class in neuro_sdk*), 25

I

id (*neuro_sdk.Bucket attribute*), 57

id (*neuro_sdk.Disk attribute*), 48
id (*neuro_sdk.JobDescription attribute*), 25
id (*neuro_sdk.PersistentBucketCredentials attribute*), 58
id (*neuro_sdk.ServiceAccount attribute*), 50
image (*neuro_sdk.Container attribute*), 24
ImageProgressPull (*class in neuro_sdk*), 42
ImageProgressPush (*class in neuro_sdk*), 42
ImageProgressStep (*class in neuro_sdk*), 42
Images (*class in neuro_sdk*), 40
images (*neuro_sdk.Client attribute*), 14
import_external() (*neuro_sdk.Buckets method*), 50
internal_hostname (*neuro_sdk.JobDescription attribute*), 26
internal_hostname_named (*neuro_sdk.JobDescription attribute*), 26
is_config_present (*neuro_sdk.Factory attribute*), 12
is_dir (*neuro_sdk.StorageProgressDelete attribute*), 39
is_dir() (*neuro_sdk.BucketEntry method*), 59
is_dir() (*neuro_sdk.FileStatus method*), 37
is_file() (*neuro_sdk.BucketEntry method*), 59
is_file() (*neuro_sdk.FileStatus method*), 37
is_finished (*neuro_sdk.JobStatus attribute*), 28
is_pending (*neuro_sdk.JobStatus attribute*), 27
is_preemptible (*neuro_sdk.Preset attribute*), 17
is_running (*neuro_sdk.JobStatus attribute*), 28
items() (*neuro_sdk.JobStatus method*), 27

J

JobDescription (*class in neuro_sdk*), 25
JobPriority (*class in neuro_sdk*), 27
JobRestartPolicy (*class in neuro_sdk*), 26
Jobs (*class in neuro_sdk*), 18
jobs (*neuro_sdk.Client attribute*), 14
JobStatus (*class in neuro_sdk*), 27
JobStatusHistory (*class in neuro_sdk*), 28
JobStatusItem (*class in neuro_sdk*), 28
JobTelemetry (*class in neuro_sdk*), 29

K

key (*neuro_sdk.BucketEntry attribute*), 59
key (*neuro_sdk.Secret attribute*), 47
kill() (*neuro_sdk.Jobs method*), 19

L

last_usage (*neuro_sdk.Disk attribute*), 48
layer_id (*neuro_sdk.ImageProgressStep attribute*), 42
leave() (*neuro_sdk.AbstractRecursiveFileProgress method*), 37
life_span (*neuro_sdk.JobDescription attribute*), 26
list() (*neuro_sdk.Buckets method*), 50
list() (*neuro_sdk.Disks method*), 47
list() (*neuro_sdk.Images method*), 41
list() (*neuro_sdk.Jobs method*), 19
list() (*neuro_sdk.Secrets method*), 46
list() (*neuro_sdk.ServiceAccounts method*), 49
list() (*neuro_sdk.Storage method*), 32
list_blobs() (*neuro_sdk.Buckets method*), 53
local_image() (*neuro_sdk.Parser method*), 60
LocalImage (*class in neuro_sdk*), 42
login() (*in module neuro_sdk*), 11
login() (*neuro_sdk.Factory method*), 12
login_headless() (*neuro_sdk.Factory method*), 13
login_with_headless() (*in module neuro_sdk*), 11
login_with_passed_config() (*neuro_sdk.Factory method*), 13
login_with_token() (*in module neuro_sdk*), 11
login_with_token() (*neuro_sdk.Factory method*), 13
logout() (*in module neuro_sdk*), 12
logout() (*neuro_sdk.Factory method*), 14
LOW (*neuro_sdk.JobPriority attribute*), 27

M

make_signed_url() (*neuro_sdk.Buckets method*), 56
MANAGE (*neuro_sdk.Action attribute*), 45
memory (*neuro_sdk.JobTelemetry attribute*), 29
memory_mb (*neuro_sdk.Preset attribute*), 17
memory_mb (*neuro_sdk.Resources attribute*), 30
Message (*class in neuro_sdk*), 30
message (*neuro_sdk.ImageProgressStep attribute*), 42
message (*neuro_sdk.StorageProgressFail attribute*), 39
MINIO (*neuro_sdk.Bucket.Provider attribute*), 58
mkdir() (*neuro_sdk.Storage method*), 32
modification_time (*neuro_sdk.FileStatus attribute*), 36
modified_at (*neuro_sdk.BucketEntry attribute*), 59
module
    neuro_sdk, 11
monitoring_url (*neuro_sdk.Cluster attribute*), 17
monitoring_url (*neuro_sdk.Config attribute*), 16
mv() (*neuro_sdk.Storage method*), 33

N

name (*neuro_sdk.Bucket attribute*), 57
name (*neuro_sdk.BucketEntry attribute*), 59
name (*neuro_sdk.Cluster attribute*), 17
name (*neuro_sdk.Disk attribute*), 48
name (*neuro_sdk.FileStatus attribute*), 36
name (*neuro_sdk.JobDescription attribute*), 25
name (*neuro_sdk.LocalImage attribute*), 42
name (*neuro_sdk.PersistentBucketCredentials attribute*), 58
name (*neuro_sdk.RemoteImage attribute*), 43
name (*neuro_sdk.ServiceAccount attribute*), 50
name (*neuro_sdk.Tag attribute*), 44
neuro_sdk
    module, 11
NEVER (*neuro_sdk.JobRestartPolicy attribute*), 26

new_external_image() (*neuro_sdk.RemoteImage class method*), 43
new_neuro_image() (*neuro_sdk.RemoteImage class method*), 43
NORMAL (*neuro_sdk.JobPriority attribute*), 27
normalize_uri() (*neuro_sdk.Parser method*), 61

O

object_count (*neuro_sdk.BucketUsage attribute*), 60
ON_FAILURE (*neuro_sdk.JobRestartPolicy attribute*), 26
open() (*neuro_sdk.Storage method*), 34
org_name (*neuro_sdk.Bucket attribute*), 57
org_name (*neuro_sdk.Config attribute*), 15
org_name (*neuro_sdk.Disk attribute*), 48
org_name (*neuro_sdk.RemoteImage attribute*), 43
org_name (*neuro_sdk.Secret attribute*), 47
owner (*neuro_sdk.Bucket attribute*), 57
owner (*neuro_sdk.Disk attribute*), 48
owner (*neuro_sdk.JobDescription attribute*), 25
owner (*neuro_sdk.PersistentBucketCredentials attribute*), 58
owner (*neuro_sdk.RemoteImage attribute*), 43
owner (*neuro_sdk.Secret attribute*), 47
owner (*neuro_sdk.ServiceAccount attribute*), 50

P

parse (*neuro_sdk.Client attribute*), 14
Parser (*class in neuro_sdk*), 60
pass_config (*neuro_sdk.JobDescription attribute*), 25
path (*neuro_sdk.Factory attribute*), 12
path (*neuro_sdk.FileStatus attribute*), 36
path_to_uri() (*neuro_sdk.Parser method*), 61
PENDING (*neuro_sdk.Disk.Status attribute*), 49
PENDING (*neuro_sdk.JobStatus attribute*), 27
Permission (*class in neuro_sdk*), 46
permission (*neuro_sdk.FileStatus attribute*), 36
permission (*neuro_sdk.Share attribute*), 45
persistent_credentials_create() (*neuro_sdk.Buckets method*), 56
persistent_credentials_get() (*neuro_sdk.Buckets method*), 57
persistent_credentials_list() (*neuro_sdk.Buckets method*), 56
persistent_credentials_rm() (*neuro_sdk.Buckets method*), 57
PersistentBucketCredentials (*class in neuro_sdk*), 58
PluginManager (*class in neuro_sdk*), 62
port (*neuro_sdk.HTTPPort attribute*), 25
port_forward() (*neuro_sdk.Jobs method*), 21
preemptible_node (*neuro_sdk.JobDescription attribute*), 25
Preset (*class in neuro_sdk*), 17
presets (*neuro_sdk.Client attribute*), 14
presets (*neuro_sdk.Cluster attribute*), 17
presets (*neuro_sdk.Config attribute*), 15
priority (*neuro_sdk.JobDescription attribute*), 26
privileged (*neuro_sdk.JobDescription attribute*), 25
provider (*neuro_sdk.Bucket attribute*), 58
provider (*neuro_sdk.BucketCredentials attribute*), 58
pull() (*neuro_sdk.AbstractDockerImageProgress method*), 41
pull() (*neuro_sdk.Images method*), 40
push() (*neuro_sdk.AbstractDockerImageProgress method*), 41
push() (*neuro_sdk.Images method*), 40
put_blob() (*neuro_sdk.Buckets method*), 52

R

READ (*neuro_sdk.Action attribute*), 45
read_only (*neuro_sdk.DiskVolume attribute*), 31
read_only (*neuro_sdk.PersistentBucketCredentials attribute*), 58
read_only (*neuro_sdk.Volume attribute*), 31
read_out() (*neuro_sdk.StdStream method*), 30
READY (*neuro_sdk.Disk.Status attribute*), 49
reason (*neuro_sdk.JobStatusHistory attribute*), 28
reason (*neuro_sdk.JobStatusItem attribute*), 28
registry (*neuro_sdk.RemoteImage attribute*), 43
registry_url (*neuro_sdk.Cluster attribute*), 17
registry_url (*neuro_sdk.Config attribute*), 16
remote_image() (*neuro_sdk.Parser method*), 60
RemoteImage (*class in neuro_sdk*), 43
remove() (*neuro_sdk.Users method*), 45
request_tmp_credentials() (*neuro_sdk.Buckets method*), 51
requires_auth (*neuro_sdk.HTTPPort attribute*), 25
Resources (*class in neuro_sdk*), 30
resources (*neuro_sdk.Container attribute*), 24
restarts (*neuro_sdk.JobStatusHistory attribute*), 29
revoke() (*neuro_sdk.Users method*), 45
rm() (*neuro_sdk.Buckets method*), 51
rm() (*neuro_sdk.Disks method*), 48
rm() (*neuro_sdk.Images method*), 40
rm() (*neuro_sdk.Secrets method*), 46
rm() (*neuro_sdk.ServiceAccounts method*), 49
rm() (*neuro_sdk.Storage method*), 33
role (*neuro_sdk.ServiceAccount attribute*), 50
role_deleted (*neuro_sdk.ServiceAccount attribute*), 50
run() (*neuro_sdk.Jobs method*), 21
RUNNING (*neuro_sdk.JobStatus attribute*), 27

S

schedule_timeout (*neuro_sdk.JobDescription attribute*), 26
scheduler_enabled (*neuro_sdk.JobDescription attribute*), 25
Secret (*class in neuro_sdk*), 47

secret_env (*neuro_sdk.Container attribute*), 24
secret_env (*neuro_sdk.EnvParseResult attribute*), 62
secret_files (*neuro_sdk.Container attribute*), 24
secret_files (*neuro_sdk.VolumeParseResult attribute*), 62
secret_uri (*neuro_sdk.SecretFile attribute*), 31
SecretFile (*class in neuro_sdk*), 31
Secrets (*class in neuro_sdk*), 46
secrets (*neuro_sdk.Client attribute*), 14
send_signal() (*neuro_sdk.Jobs method*), 23
ServiceAccount (*class in neuro_sdk*), 50
ServiceAccounts (*class in neuro_sdk*), 49
set_public_access() (*neuro_sdk.Buckets method*), 52
Share (*class in neuro_sdk*), 45
share() (*neuro_sdk.Users method*), 44
shm (*neuro_sdk.Resources attribute*), 30
size (*neuro_sdk.BucketEntry attribute*), 59
size (*neuro_sdk.FileStatus attribute*), 36
size (*neuro_sdk.StorageProgressComplete attribute*), 38
size (*neuro_sdk.StorageProgressStart attribute*), 38
size (*neuro_sdk.StorageProgressStep attribute*), 38
size (*neuro_sdk.Tag attribute*), 44
size() (*neuro_sdk.Images method*), 41
src (*neuro_sdk.ImageProgressPull attribute*), 42
src (*neuro_sdk.ImageProgressPush attribute*), 42
src (*neuro_sdk.StorageProgressComplete attribute*), 38
src (*neuro_sdk.StorageProgressEnterDir attribute*), 39
src (*neuro_sdk.StorageProgressFail attribute*), 39
src (*neuro_sdk.StorageProgressLeaveDir attribute*), 39
src (*neuro_sdk.StorageProgressStart attribute*), 38
src (*neuro_sdk.StorageProgressStep attribute*), 38
ssh_server (*neuro_sdk.JobDescription attribute*), 26
start() (*neuro_sdk.AbstractFileProgress method*), 37
start() (*neuro_sdk.Jobs method*), 21
started_at (*neuro_sdk.JobStatusHistory attribute*), 29
stat() (*neuro_sdk.Storage method*), 33
status (*neuro_sdk.Disk attribute*), 48
status (*neuro_sdk.JobDescription attribute*), 25
status (*neuro_sdk.JobStatusHistory attribute*), 28
status (*neuro_sdk.JobStatusItem attribute*), 28
status() (*neuro_sdk.Jobs method*), 23
StdStream (*class in neuro_sdk*), 30
step() (*neuro_sdk.AbstractDockerImageProgress method*), 41
step() (*neuro_sdk.AbstractFileProgress method*), 37
Storage (*class in neuro_sdk*), 32
storage (*neuro_sdk.Client attribute*), 14
storage (*neuro_sdk.Disk attribute*), 48
storage_uri (*neuro_sdk.Volume attribute*), 31
storage_url (*neuro_sdk.Cluster attribute*), 17
storage_url (*neuro_sdk.Config attribute*), 16
StorageProgressComplete (*class in neuro_sdk*), 38
StorageProgressDelete (*class in neuro_sdk*), 39
StorageProgressEnterDir (*class in neuro_sdk*), 39
StorageProgressFail (*class in neuro_sdk*), 39
StorageProgressLeaveDir (*class in neuro_sdk*), 39
StorageProgressStart (*class in neuro_sdk*), 38
StorageProgressStep (*class in neuro_sdk*), 38
str_to_uri() (*neuro_sdk.Parser method*), 61
SUCCEEDED (*neuro_sdk.JobStatus attribute*), 27
SUSPENDED (*neuro_sdk.JobStatus attribute*), 27
switch_cluster() (*neuro_sdk.Config method*), 15
switch_org() (*neuro_sdk.Config method*), 16

T

Tag (*class in neuro_sdk*), 43
tag (*neuro_sdk.LocalImage attribute*), 42
tag (*neuro_sdk.RemoteImage attribute*), 43
tag_info() (*neuro_sdk.Images method*), 41
tags (*neuro_sdk.JobDescription attribute*), 25
tags() (*neuro_sdk.Images method*), 41
timeout_unused (*neuro_sdk.Disk attribute*), 48
timestamp (*neuro_sdk.JobTelemetry attribute*), 29
token() (*neuro_sdk.Config method*), 17
top() (*neuro_sdk.Jobs method*), 23
total (*neuro_sdk.DiskUsageInfo attribute*), 39
total_bytes (*neuro_sdk.BucketUsage attribute*), 60
tpu_software_version (*neuro_sdk.Preset attribute*), 18
tpu_software_version (*neuro_sdk.Resources attribute*), 30
tpu_type (*neuro_sdk.Preset attribute*), 18
tpu_type (*neuro_sdk.Resources attribute*), 30
transition_time (*neuro_sdk.JobStatusItem attribute*), 28
transitions (*neuro_sdk.JobStatusHistory attribute*), 29
tty, **63**
type (*neuro_sdk.FileStatus attribute*), 37

U

UNKNOWN (*neuro_sdk.JobStatus attribute*), 27
upload_dir() (*neuro_sdk.Buckets method*), 54
upload_dir() (*neuro_sdk.Storage method*), 35
upload_file() (*neuro_sdk.Buckets method*), 54
upload_file() (*neuro_sdk.Storage method*), 36
uri (*neuro_sdk.Bucket attribute*), 57
uri (*neuro_sdk.BucketEntry attribute*), 59
uri (*neuro_sdk.Disk attribute*), 48
uri (*neuro_sdk.Permission attribute*), 46
uri (*neuro_sdk.Secret attribute*), 47
uri (*neuro_sdk.StorageProgressDelete attribute*), 39
uri_to_path() (*neuro_sdk.Parser method*), 61
uri_to_str() (*neuro_sdk.Parser method*), 61
used (*neuro_sdk.DiskUsageInfo attribute*), 39
used_bytes (*neuro_sdk.Disk attribute*), 48
user (*neuro_sdk.Share attribute*), 45
username (*neuro_sdk.Client attribute*), 14
username (*neuro_sdk.Config attribute*), 15

Users (*class in neuro_sdk*), 44
users (*neuro_sdk.Client attribute*), 14
users_url (*neuro_sdk.Cluster attribute*), 17

V

Volume (*class in neuro_sdk*), 31
volume() (*neuro_sdk.Parser method*), 60
VolumeParseResult (*class in neuro_sdk*), 62
volumes (*neuro_sdk.Container attribute*), 24
volumes (*neuro_sdk.VolumeParseResult attribute*), 62
volumes() (*neuro_sdk.Parser method*), 61

W

with_tag() (*neuro_sdk.RemoteImage method*), 43
WRITE (*neuro_sdk.Action attribute*), 45
write() (*neuro_sdk.Storage method*), 34
write_in() (*neuro_sdk.StdStream method*), 30