Neptune client

Release 0.2.9

team neptune.ml

Aug 20, 2019

TUTORIALS

Neptune is a collaboration platform for data science / machine learning teams that focuses on three areas:

- **Track**: all metrics and outputs in your data science or machine learning project. It can be model training curves, visualizations, input data, calculated features and so on.
- **Organize**: automatically transform tracked data into a knowledge repository.
- **Collaborate**: share, compare and discuss your work across data science project.

CHAPTER

ONE

WHAT IS NEPTUNE CLIENT?

Neptune client is open source Python library that allows Users to integrate their Python scripts with Neptune.

Note: Make sure to register to Neptune, to use it.

CHAPTER TWO

INSTALLATION

```
pip install neptune-client
```

Once installed, `import neptune` in your code to use it.

CHAPTER

THREE

EXAMPLE

```
import neptune

neptune.init('shared/onboarding')
with neptune.create_experiment(name='simple_example'):
    neptune.append_tag('minimal-example')
    n = 117
    for i in range(1, n):
        neptune.send_metric('iteration', i)
        neptune.send_metric('loss', 1/i**0.5)
    neptune.set_property('n_iterations', n)
```

Example above creates Neptune experiment in the project: *shared/onboarding* and logs *iteration* and *loss* metrics to Neptune in real time. It also presents common use case for Neptune client, that is tracking progress of machine learning experiments.

3.1 Minimal example

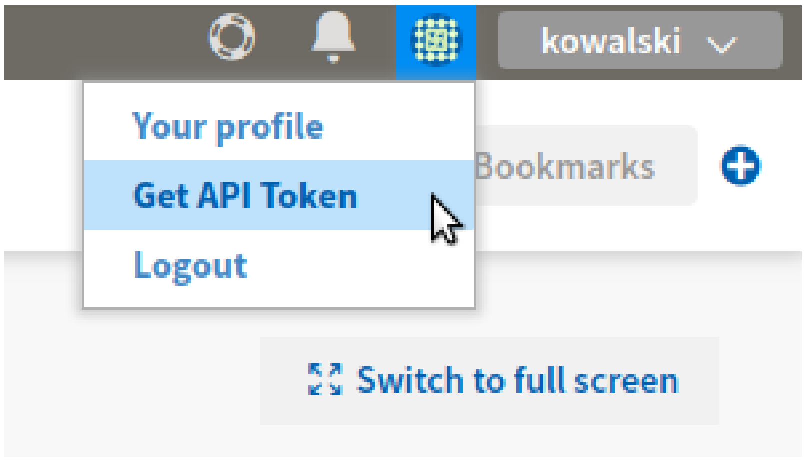
Below is the smallest possible example that follows the theme: *from zero to first Neptune experiment*.

3.1.1 Register

Go here: https://neptune.ml/register *(registration is free of charge)*.

3.1.2 Copy API token

`NEPTUNE_API_TOKEN` is located under your User menu (top right side of the screen, like on the image below):



Assign it to the bash environment variable:

```
export NEPTUNE_API_TOKEN='YOUR_LONG_API_TOKEN'
```

or append this line to your `~/.bashrc` or `~/.bash_profile` files **(recommended)**.

> **Warning:** Always keep your API token secret - it is like password to the application. It is recommended to append "export NEPTUNE_API_TOKEN='YOUR_LONG_API_TOKEN'" line to your `~/.bashrc` or `~/.bash_profile` files.

3.1.3 Install neptune-client

```
pip install neptune-client
```

Install psutil to see hardware monitoring charts:

```
pip3 install psutil
```

(please check psutil documentation *in case of installation problems)*

3.1.4 Run Python script

Save script below as `start.py` and run it like any other Python file: `python start.py`. Will see link to the experiment printed to the standard output.

```
import neptune

# pick project, provide API token
neptune.init('USERNAME/PROJECT_NAME')

# create experiment
neptune.create_experiment()

# send some metrics
n = 117
for i in range(1, n):
    neptune.send_metric('iteration', i)
```

(continues on next page)

(continued from previous page)

```
    neptune.send_metric('loss', 1/i**0.5)

neptune.set_property('n_iterations', n)
neptune.stop()
```

Congrats! You just ran your first Neptune experiment and checked results online.

Note: What did you just learn? Few concepts:

- how to run Neptune experiment
- how to track it online
- how to use basic Neptune client features, like *create_experiment()* and *send_metric()*

3.2 Session and Experiment

Session and experiment are two core concepts behind neptune-client. This tutorial guides you through them and explains how to work with them.

3.2.1 Session

In the first tutorial, as you remember, we initialized Neptune using *neptune.init*:

```
import neptune

neptune.init('USERNAME/PROJECT_NAME')
...
```

However, full definition of the *neptune.init* is like this:

```
neptune.init(project_qualified_name='USERNAME/PROJECT_NAME',
             api_token='YOUR_LONG_API_TOKEN')
```

- `project_qualified_name` - this is *USERNAME/PROJECT_NAME*, where first component is organization name and second is project name (as you created in the Neptune web application).
- `api_token` - User can explicitly paste `NEPTUNE_API_TOKEN` here, however, **it is not recommended**. This method first look for environment variable, then value passed here. Note that this value overwrites environment variable.

Note: If you have your API token stored in the `NEPTUNE_API_TOKEN` environment variable you can leave the `api_token` argument empty.

That is not the only way of doing it but does make things simpler. If you want to have more control you can explicitly start neptune session:

```
from neptune.sessions import Session

session = Session(api_token='YOUR_LONG_API_TOKEN')
```

The session object lazily contains all of the projects that you have access too. You can fetch the project on which you want to work on by running:

```
project = session.get_project(project_qualified_name='USERNAME/PROJECT_NAME')
```

And create a new experiment in that project.

```
experiment = project.create_experiment()
```

Returned `experiment` lets you invoke all methods that you know from the previous tutorial, for example:

```
experiment = project.create_experiment()

experiment.send_metric('iteration', i)
experiment.send_metric('loss', 1/i**0.5)
experiment.set_property('n_iterations', 117)
```

3.2.2 Experiment

Let's dive into the **create_experiment** method and what you can track with it. As you remember in the minimal example we started an experiment, logged something to it, and stopped it:

```
neptune.create_experiment()
neptune.send_metric('auc', 0.93)
neptune.stop()
```

You can make it cleaner and create your experiments in **with statement** blocks:

```
with neptune.create_experiment() as exp:
    exp.send_metric('auc', 0.93)
```

By doing that you will never forget to stop your experiments. We recommend you use this option. Also, if you are creating more than one experiment, this approach keeps things civil.

Ok, now that we know how to start and stop experiments let's see what happens in the app when you actually run it.

With every **create_experiment** a new record is added to Neptune with a state *running*. When you run **stop** on your experiment, either explicitly or implicitly, the state is changed to *succeeded*.

3.3 Advanced example

This example uses Get Started with TensorFlow as a base. It contains more features that neptune-client has to offer and put them in single script. Specifically, you will see several methods in action:

- `send_text()`
- `send_metric()`
- `send_artifact()`
- `append_tag()`
- `send_text()`
- `set_property()`

Copy it and save as *example.py*, then run it as usual: `python example.py`. In this tutorial we make use of the public `NEPTUNE_API_TOKEN` of the public user Neptuner. Thus, when started you can see your experiment at the top of experiments view.

```
from hashlib import sha1

import keras
import neptune
from keras import backend as K
from keras.callbacks import Callback

PARAMS = {'lr': 0.0001,
          'dropout': 0.2,
          'batch_size': 64,
          'optimizer': 'adam',
          'loss': 'sparse_categorical_crossentropy',
          'metrics': 'accuracy',
          'n_epochs': 5,
          }

# prepare Keras callback to track training progress in Neptune
class NeptuneMonitor(Callback):
    def __init__(self, neptune_experiment, n_batch):
        super().__init__()
        self.exp = neptune_experiment
        self.n = n_batch
        self.current_epoch = 0

    def on_batch_end(self, batch, logs=None):
        x = (self.current_epoch * self.n) + batch
        self.exp.send_metric(channel_name='batch end accuracy', x=x, y=logs['acc'])
        self.exp.send_metric(channel_name='batch end loss', x=x, y=logs['loss'])

    def on_epoch_end(self, epoch, logs=None):
        self.exp.send_metric('epoch end accuracy', logs['acc'])
        self.exp.send_metric('epoch end loss', logs['loss'])

        innovative_metric = logs['acc'] - 2 * logs['loss']
        self.exp.send_metric(channel_name='innovative_metric', x=epoch, y=innovative_
↪metric)

        msg_acc = 'End of epoch {}, accuracy is {:.4f}'.format(epoch, logs['acc'])
        self.exp.send_text(channel_name='accuracy information', x=epoch, y=msg_acc)

        msg_loss = 'End of epoch {}, categorical crossentropy loss is {:.4f}'.
↪format(epoch, logs['loss'])
        self.exp.send_text(channel_name='loss information', x=epoch, y=msg_loss)

        self.current_epoch += 1

# retrieve project
project = neptune.Session(
↪'eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vdWkubmVwdHVuZS5tbCIsImFwaV9rZXkiOiJiNzA2YmM4Zi03NmY5LTRjMmUtOTM5Z
↪')\
    .get_project('shared/onboarding')

# create context with 'npt_exp', so you do not need to remember to close it at the end
with project.create_experiment(name='neural-net-mnist',
```

(continues on next page)

(continued from previous page)

```
                              params=PARAMS,
                              description='neural net trained on MNIST',
                              upload_source_files=['example.py']) as npt_exp:

    # prepare data
    mnist = keras.datasets.mnist
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    x_train, x_test = x_train / 255.0, x_test / 255.0

    # calculate number of batches per epoch and track it in Neptune
    n_batches = x_train.shape[0] // npt_exp.get_parameters()['batch_size'] + 1
    npt_exp.set_property('n_batches', n_batches)

    # calculate train / test data hash and track it in Neptune
    train_sha = sha1(x_train).hexdigest()
    test_sha = sha1(x_test).hexdigest()
    npt_exp.send_text('train_version', train_sha)
    npt_exp.send_text('test_version', test_sha)

    # prepare model that use dropout parameter from Neptune
    model = keras.models.Sequential([
        keras.layers.Flatten(),
        keras.layers.Dense(512, activation=K.relu),
        keras.layers.Dropout(npt_exp.get_parameters()['dropout']),
        keras.layers.Dense(10, activation=K.softmax)
    ])

    # compile model using use parameters from Neptune
    model.compile(optimizer=npt_exp.get_parameters()['optimizer'],
                  loss=npt_exp.get_parameters()['loss'],
                  metrics=[npt_exp.get_parameters()['metrics']])

    # fit the model to data, using NeptuneMonitor callback
    model.fit(x_train, y_train,
              epochs=PARAMS['n_epochs'],
              batch_size=PARAMS['batch_size'],
              callbacks=[NeptuneMonitor(npt_exp, n_batches)])

    # evaluate model on test data and track it in Neptune
    names = model.metrics_names
    values = model.evaluate(x_test, y_test)
    npt_exp.set_property(names[0], values[0])
    npt_exp.set_property(names[1], values[1])

    # save model in Neptune
    model.save_weights('model_weights.h5')
    npt_exp.send_artifact('model_weights.h5')
    npt_exp.append_tag('large lr')
    npt_exp.append_tag('compare')
```

Run this code and observe results online.

3.4 Neptune

`neptune.`**`append_tag`**(*tag*, **tags*)

Append tag(s) to the experiment on the top of experiments view.

Alias for: *`append_tag()`*

`neptune.`**`append_tags`**(*tag*, **tags*)

Append tag(s) to the experiment on the top of experiments view.

Alias for: *`append_tags()`*

`neptune.`**`create_experiment`**(*name=None*, *description=None*, *params=None*, *properties=None*, *tags=None*, *upload_source_files=None*, *abort_callback=None*, *logger=None*, *upload_stdout=True*, *upload_stderr=True*, *send_hardware_metrics=True*, *run_monitoring_thread=True*, *handle_uncaught_exceptions=True*, *git_info=None*, *hostname=None*, *notebook_id=None*)

Create and start Neptune experiment.

Alias for: *`create_experiment()`*

`neptune.`**`init`**(*project_qualified_name=None*, *api_token=None*, *proxies=None*)

Initialize Neptune client library to work with specific project.

Authorize user, sets value of global variable `project` to *`Project`* object that can be used to create or list experiments, notebooks, etc.

Parameters

- **`project_qualified_name`** (`str`, optional, default is `None`) – Qualified name of a project in a form of `namespace/project_name`. If `None`, the value of `NEPTUNE_PROJECT` environment variable will be taken.
- **`api_token`** (`str`, optional, default is `None`) – User's API token. If `None`, the value of `NEPTUNE_API_TOKEN` environment variable will be taken.
- **`proxies`** (`str`, optional, default is `None`) – Argument passed to HTTP calls made via the Requests library. For more information see their proxies section.

Note: It is strongly recommended to use `NEPTUNE_API_TOKEN` environment variable rather than placing your API token in plain text in your source code.

Returns *`Project`* object that is used to create or list experiments, notebooks, etc.

Raises

- **`MissingApiToken`** – When `api_token` is None and `NEPTUNE_API_TOKEN` environment variable was not set.
- **`MissingProjectQualifiedName`** – When `project_qualified_name` is None and `NEPTUNE_PROJECT` environment variable was not set.
- **`InvalidApiKey`** – When given `api_token` is malformed.
- **`Unauthorized`** – When given `api_token` is invalid.

Examples

```
# minimal invoke
neptune.init()

# specifying project name
neptune.init('jack/sandbox')
```

`neptune.`**`log_artifact`**`(`*artifact*, *destination=None*`)`
Save an artifact (file) in experiment storage.

Alias for `log_artifact()`

`neptune.`**`log_image`**`(`*log_name*, *x*, *y=None*, *name=None*, *description=None*, *timestamp=None*`)`
Log image data in Neptune.

Alias for `log_image()`

`neptune.`**`log_metric`**`(`*log_name*, *x*, *y=None*, *timestamp=None*`)`
Log metrics (numeric values) in Neptune.

Alias for `log_metric()`

`neptune.`**`log_text`**`(`*log_name*, *x*, *y=None*, *timestamp=None*`)`
Log text data in Neptune.

Alias for `log_text()`

`neptune.`**`remove_property`**`(`*key*`)`
Removes a property with given key.

Alias for: `remove_property()`

`neptune.`**`remove_tag`**`(`*tag*`)`
Removes single tag from experiment.

Alias for: `remove_tag()`

`neptune.`**`send_artifact`**`(`*artifact*, *destination=None*`)`
Save an artifact (file) in experiment storage.

Alias for `log_artifact()`

`neptune.`**`send_image`**`(`*channel_name*, *x*, *y=None*, *name=None*, *description=None*, *timestamp=None*`)`
Log image data in Neptune.

Alias for `log_image()`

`neptune.`**`send_metric`**`(`*channel_name*, *x*, *y=None*, *timestamp=None*`)`
Log metrics (numeric values) in Neptune.

Alias for `log_metric()`

`neptune.`**`send_text`**`(`*channel_name*, *x*, *y=None*, *timestamp=None*`)`
Log text data in Neptune.

Alias for `log_text()`

`neptune.`**`set_project`**`(`*project_qualified_name*`)`
Setups Neptune client library to work with specific project.

Sets value of global variable `project` to `Project` object that can be used to create or list experiments, notebooks, etc.

If Neptune client library was not previously initialized via *`init()`* call it will be initialized with API token taken from `NEPTUNE_API_TOKEN` environment variable.

Parameters **`project_qualified_name`** (`str`) – Qualified name of a project in a form of `namespace/project_name`.

Returns *`Project`* object that is used to create or list experiments, notebooks, etc.

Raises **`MissingApiToken`** – When library was not initialized previously by `init` call and `NEPTUNE_API_TOKEN` environment variable is not set.

Examples

```
# minimal invoke
neptune.set_project('jack/sandbox')
```

`neptune.`**`set_property`**(*key*, *value*)

Set *key-value* pair as an experiment property.

If property with given `key` does not exist, it adds a new one.

Alias for: *`set_property()`*

`neptune.`**`stop`**(*traceback=None*)

Marks experiment as finished (succeeded or failed).

Alias for *`stop()`*

3.5 Session

`class` `neptune.sessions.`**`Session`**(*api_token=None*, *proxies=None*)

Bases: `object`

A class for running communication with Neptune.

In order to query Neptune experiments you need to instantiate this object first.

Parameters

- **`api_token`** (`str`, optional, default is `None`) – User's API token. If `None`, the value of `NEPTUNE_API_TOKEN` environment variable will be taken.
- **`proxies`** (`str`, optional, default is `None`) – Argument passed to HTTP calls made via the Requests library. For more information see their proxies section.

Examples

Create session and pass 'api_token'

```
from neptune.sessions import Session
session = Session(api_token='YOUR_NEPTUNE_API_TOKEN')
```

Create session, assuming you have created an environment variable 'NEPTUNE_API_TOKEN'

```
from neptune.sessions import Session
session = Session()
```

get_project(*project_qualified_name*)

Get a project with given `project_qualified_name`.

In order to access experiments data one needs to get a *`Project`* object first. This method gives you the ability to do that.

Parameters **`project_qualified_name`** (`str`) – Qualified name of a project in a form of `namespace/project_name`.

Returns *`Project`* object.

Raise: `ProjectNotFound`: When a project with given name does not exist.

Examples

```
# Create a Session instance
from neptune.sessions import Session
session = Session()

# Get a project by it's ``project_qualified_name``:
my_project = session.get_project('namespace/project_name')
```

get_projects(*namespace*)

Get all projects that you have permissions to see in given organization

This method gets you all available projects names and their corresponding *`Project`* objects.

Both private and public projects may be returned for the organization. If you have role in private project, it is included.

You can retrieve all the public projects that belong to any user or organization, as long as you know their username or organization name.

Parameters **`namespace`** (`str`) – It can either be name of the organization or username.

Returns

`OrderedDict`

keys are `project_qualified_name` that is: *'organization/project_name'*

values are corresponding *`Project`* objects.

Raises **`NamespaceNotFound`** – When the given namespace does not exist.

Examples

```
# create Session
from neptune.sessions import Session
session = Session()

# Now, you can list all the projects available for a selected namespace.
# You can use `YOUR_NAMESPACE` which is your organization or user name.
# You can also list public projects created by other organizations.
# For example you can use the `neptune-ml` namespace.

session.get_projects('neptune-ml')
```

(continues on next page)

(continued from previous page)

```
# Example output:
# OrderedDict([('neptune-ml/credit-default-prediction',
#               Project(neptune-ml/credit-default-prediction)),
#              ('neptune-ml/GStore-Customer-Revenue-Prediction',
#               Project(neptune-ml/GStore-Customer-Revenue-Prediction)),
#              ('neptune-ml/human-protein-atlas',
#               Project(neptune-ml/human-protein-atlas)),
#              ('neptune-ml/Ships',
#               Project(neptune-ml/Ships)),
#              ('neptune-ml/Mapping-Challenge',
#               Project(neptune-ml/Mapping-Challenge))
#             ])
```

3.6 Project

class neptune.projects.**Project**(*client*, *internal_id*, *namespace*, *name*)

Bases: `object`

A class for storing information and managing Neptune project.

Parameters

- **client** (`Client`, required) – Client object.
- **internal_id** (`str`, required) – UUID of the project.
- **namespace** (`str`, required) – It can either be your organization or user name.
- **name** (`str`, required) – project name.

Note: `namespace` and `name` joined together form `project_qualified_name`.

create_experiment(*name=None*, *description=None*, *params=None*, *properties=None*, *tags=None*, *upload_source_files=None*, *abort_callback=None*, *logger=None*, *upload_stdout=True*, *upload_stderr=True*, *send_hardware_metrics=True*, *run_monitoring_thread=True*, *handle_uncaught_exceptions=True*, *git_info=None*, *hostname=None*, *notebook_id=None*)

Create and start Neptune experiment.

Create experiment, set its status to *running* and append it to the top of the experiments view. All parameters are optional, hence minimal invocation: `neptune.create_experiment()`.

Parameters

- **name** (`str`, optional, default is `'Untitled'`) – Editable name of the experiment. Name is displayed in the experiment's *Details* (*Metadata* section) and in *experiments view* as a column.
- **description** (`str`, optional, default is `''`) – Editable description of the experiment. Description is displayed in the experiment's *Details* (*Metadata* section) and can be displayed in the *experiments view* as a column.
- **params** (`dict`, optional, default is `{}`) – Parameters of the experiment. After experiment creation `params` are read-only (see: `get_parameters()`). Parameters are displayed

in the experiment's *Details* (*Parameters* section) and each key-value pair can be viewed in *experiments view* as a column.

- **properties** (dict, optional, default is {}) – Properties of the experiment. They are editable after experiment is created. Properties are displayed in the experiment's *Details* (*Properties* section) and each key-value pair can be viewed in *experiments view* as a column.
- **tags** (list, optional, default is []) – Must be list of str. Tags of the experiment. They are editable after experiment is created (see: *append_tag()* and *remove_tag()*). Tags are displayed in the experiment's *Details* (*Metadata* section) and can be viewed in *experiments view* as a column.
- **upload_source_files** (list, optional, default is ['main.py']) –

  Where *'main.py'* is Python file from which experiment was created (name *'main.py'* is just an example here). Must be list of str. Uploaded sources are displayed in the experiment's *Source code* tab.

  Pass empty list ([]) to upload no files.
- **abort_callback** (callable, optional, default is None) – Callback that defines how *abort experiment* action in the Web application should work. Actual behavior depends on your setup:
  - (default) If abort_callback=None and psutil is installed, then current process and it's children are aborted by sending *SIGTERM*. If, after grace period, processes are not terminated, *SIGKILL* is sent.
  - If abort_callback=None and psutil is **not** installed, then *abort experiment* action just marks experiment as *aborted* in the Web application. No action is performed on the current process.
  - If abort_callback=callable, then callable is executed when *abort experiment* action in the Web application is triggered.
- **logger** (logging.handlers or *None*, optional, default is None) – If handler to *Python logger* is passed, new experiment's *text log* (see: *log_text()*) with name *"logger"* is created. Each time *Python logger* logs new data, it is automatically sent to the *"logger"* in experiment. As a results all data from *Python logger* are in the *Logs* tab in the experiment.
- **upload_stdout** (Boolean, optional, default is True) – Whether to send stdout to experiment's *Monitoring*.
- **upload_stderr** (Boolean, optional, default is True) – Whether to send stderr to experiment's *Monitoring*.
- **send_hardware_metrics** (Boolean, optional, default is True) – Whether to send hardware monitoring logs (CPU, GPU, Memory utilization) to experiment's *Monitoring*.
- **run_monitoring_thread** (Boolean, optional, default is True) – Whether to run thread that pings Neptune server in order to determine if experiment is responsive.
- **handle_uncaught_exceptions** (Boolean, optional, default is True) – Two options True and False are possible:
  - If set to True and uncaught exception occurs, then Neptune automatically place *Traceback* in the experiment's *Details* and change experiment status to *Failed*.
  - If set to False and uncaught exception occurs, then no action is performed in the Web application. As a consequence, experiment's status is *running* or *not responding*.

- **git_info** (*GitInfo*, optional, default is None) –

  Instance of the class *GitInfo* that provides information about the git repository from which experiment was started.

  If None is passed, system attempts to automatically extract information about git repository in the following way:

  – System looks for *.git* file in the current directory and, if not found, goes up recursively until *.git* file will be found (see: *get_git_info()*).

  – If there is no git repository, then no information about git is displayed in experiment details in Neptune web application.

- **hostname** (str, optional, default is None) – If None, neptune automatically get *hostname* information. User can also set *hostname* directly by passing str.

Returns *Experiment* object that is used to manage experiment and log data to it.

Raises

- **ExperimentValidationError** – When provided arguments are invalid.
- **ExperimentLimitReached** – When experiment limit in the project has been reached.

Examples

```
# minimal invoke
neptune.create_experiment()

# explicitly return experiment object
experiment = neptune.create_experiment()

# create experiment with name and two parameters
neptune.create_experiment(name='first-pytorch-ever',
                          params={'lr': 0.0005,
                                  'dropout': 0.2})

# create experiment with name and description, and no sources files uploaded
neptune.create_experiment(name='neural-net-mnist',
                          description='neural net trained on MNIST',
                          upload_source_files=[])

# larger example
neptune.create_experiment(name='first-pytorch-ever',
                          params={'lr': 0.0005,
                                  'dropout': 0.2},
                          properties={'key1': 'value1',
                                      'key2': 17,
                                      'key3': 'other-value'},
                          description='write longer description here',
                          tags=['list-of', 'tags', 'goes-here', 'as-list-of-
↪strings'],
                          upload_source_files=['training_with_pytorch.py',
↪'net.py'])
```

create_notebook()

Create a new notebook object and return corresponding *Notebook* instance.

Returns *Notebook* object.

Examples

```
# Instantiate a session and fetch a project
project = neptune.init()

# Create a notebook in Neptune
notebook = project.create_notebook()
```

full_id
Project qualified name as `str`, for example *john/sandbox*.

get_experiments (*id=None*, *state=None*, *owner=None*, *tag=None*, *min_running_time=None*)
Retrieve list of experiments matching the specified criteria.

All parameters are optional, each of them specifies a single criterion. Only experiments matching all of the criteria will be returned.

Parameters

- **id** (`str` or `list` of `str`, optional, default is `None`) –

  An experiment id like `'SAN-1'` or list of ids like `['SAN-1', 'SAN-2']`.
  Matching any element of the list is sufficient to pass criterion.

- **state** (`str` or `list` of `str`, optional, default is `None`) –

  An experiment state like `'succeeded'` or list of states like `['succeeded', 'running']`.
  Possible values: `'running'`, `'succeeded'`, `'failed'`, `'aborted'`.
  Matching any element of the list is sufficient to pass criterion.

- **owner** (`str` or `list` of `str`, optional, default is `None`) –

  Username of the experiment owner (User who created experiment is an owner) like `'josh'` or list of owners like `['frederic', 'josh']`.
  Matching any element of the list is sufficient to pass criterion.

- **tag** (`str` or `list` of `str`, optional, default is `None`) –

  An experiment tag like `'lightGBM'` or list of tags like `['pytorch', 'cycleLR']`.
  Only experiments that have all specified tags will match this criterion.

- **min_running_time** (`int`, optional, default is `None`) – Minimum running time of an experiment in seconds, like `2000`.

Returns `list` of *`Experiment`* objects.

Examples

```
# Fetch a project
project = session.get_projects('neptune-ml')['neptune-ml/Salt-Detection']

# Get list of experiments
project.get_experiments(state=['aborted'], owner=['neyo'], min_running_
↪time=100000)

# Example output:
# [Experiment(SAL-1609),
```

(continues on next page)

(continued from previous page)

```
#   Experiment(SAL-1765),
#   Experiment(SAL-1941),
#   Experiment(SAL-1960),
#   Experiment(SAL-2025)]
```

get_leaderboard(*id=None*, *state=None*, *owner=None*, *tag=None*, *min_running_time=None*)

Fetch Neptune experiments view as pandas `DataFrame`.

returned DataFrame

In the returned `DataFrame` each *row* is an experiment and *columns* represent all system properties, numeric and text logs, parameters and properties in these experiments.

Note that, returned `DataFrame` does not contain all columns across the entire project.

Some columns may be empty, since experiments may define various logs, properties, etc.

For each log at most one (the last one) value is returned per experiment.

Text values are trimmed to 255 characters.

about parameters

All parameters are optional, each of them specifies a single criterion. Only experiments matching all of the criteria will be returned.

Parameters

- **id** (`str` or `list` of `str`, optional, default is `None`) –

  An experiment id like `'SAN-1'` or list of ids like `['SAN-1', 'SAN-2']`.
  Matching any element of the list is sufficient to pass criterion.

- **state** (`str` or `list` of `str`, optional, default is `None`) –

  An experiment state like `'succeeded'` or list of states like `['succeeded', 'running']`.
  Possible values: `'running'`, `'succeeded'`, `'failed'`, `'aborted'`.
  Matching any element of the list is sufficient to pass criterion.

- **owner** (`str` or `list` of `str`, optional, default is `None`) –

  Username of the experiment owner (User who created experiment is an owner) like `'josh'` or list of owners like `['frederic', 'josh']`.
  Matching any element of the list is sufficient to pass criterion.

- **tag** (`str` or `list` of `str`, optional, default is `None`) –

  An experiment tag like `'lightGBM'` or list of tags like `['pytorch', 'cycleLR']`.
  Only experiments that have all specified tags will match this criterion.

- **min_running_time** (`int`, optional, default is `None`) – Minimum running time of an experiment in seconds, like `2000`.

Returns `pandas.DataFrame` - Fetched Neptune experiments view.

Examples

```
# Fetch a project.
project = session.get_projects('neptune-ml')['neptune-ml/Salt-Detection']

# Get DataFrame that resembles experiment view.
project.get_leaderboard(state=['aborted'], owner=['neyo'], min_running_
↪time=100000)
```

get_members()

Retrieve a list of project members.

Returns `list` of `str` - A list of usernames of project members.

Examples

```
project = session.get_projects('neptune-ml')['neptune-ml/Salt-Detection']
project.get_members()
```

get_notebook(*notebook_id*)

Get a `Notebook` object with given `notebook_id`.

Returns `Notebook` object.

Examples

```
# Instantiate a session and fetch a project
project = neptune.init()

# Get a notebook object
notebook = project.get_notebook('d1c1b494-0620-4e54-93d5-29f4e848a51a')
```

3.7 Experiment

class `neptune.experiments.`**Experiment**(*client*, *project*, *_id*, *internal_id*)

Bases: `object`

A class for managing Neptune experiment.

Each time User creates new experiment instance of this class is created. It lets you manage experiment, `log_metric()`, `log_text()`, `log_image()`, `set_property()`, and much more.

Parameters

- **client** (`neptune.Client`) – API Client object
- **project** (`neptune.Project`) – `Project` instance
- **_id** (`str`) – Experiment short id
- **internal_id** (`str`) – internal UUID

Example

Assuming that *project* is an instance of `Project`.

```
experiment = project.create_experiment()
```

Warning: User should never create instances of this class manually. Always use: `create_experiment()`.

`append_tag`(*tag*, **tags*)

Append tag(s) to the current experiment.

Alias: `append_tags()`. Only `[a-zA-Z0-9]` and `-` (dash) characters are allowed in tags.

Parameters **`tag`** (single `str` or multiple `str` or `list` of `str`) – Tag(s) to add to the current experiment.

- If `str` is passed, singe tag is added.
- If multiple - comma separated - `str` are passed, all of them are added as tags.
- If `list` of `str` is passed, all elements of the `list` are added as tags.

Examples

```
neptune.append_tag('new-tag')  # single tag
neptune.append_tag('first-tag', 'second-tag', 'third-tag')  # few str
neptune.append_tag(['first-tag', 'second-tag', 'third-tag'])  # list of str
```

`append_tags`(*tag*, **tags*)

Append tag(s) to the current experiment.

Alias for: `append_tag()`

`download_artifact`(*filename*, *destination_dir*)

Download an artifact (file) from the experiment storage.

Download `filename` from the experiment storage and save it in `destination_dir`.

Parameters

- **`filename`** (`str`) – Name of the file to be downloaded.
- **`destination_dir`** (`str`) – The directory where the file will be downloaded.

Raises **`NotADirectory`** – When `destination_dir` is not a directory.

Examples

Assuming that *experiment* is an instance of `Experiment`.

```
experiment.download_artifact('forest_results.pkl', '/home/user/files/')
```

`get_channels`()

Alias for `get_logs()`

get_hardware_utilization()
Retrieve GPU, CPU and memory utilization data.

Get hardware utilization metrics for entire experiment as a single pandas.DataFrame object. Returned DataFrame has following columns (assuming single GPU with 0 index):

- *x_ram* - time (in milliseconds) from the experiment start,
- *y_ram* - memory usage in GB,
- *x_cpu* - time (in milliseconds) from the experiment start,
- *y_cpu* - CPU utilization percentage (0-100),
- *x_gpu_util_0* - time (in milliseconds) from the experiment start,
- *y_gpu_util_0* - GPU utilization percentage (0-100),
- *x_gpu_mem_0* - time (in milliseconds) from the experiment start,
- *y_gpu_mem_0* - GPU memory usage in GB.

If more GPUs are available they have their separate columns with appropriate indices (0, 1, 2, . . .), for example: *x_gpu_util_1*, *y_gpu_util_1*.
The returned DataFrame may contain `NaN` s if one of the metrics has more values than others.

Returns `pandas.DataFrame` - DataFrame containing the hardware utilization metrics.

Examples

The following values denote that after 3 seconds, the experiment used 16.7 GB of RAM

- *x_ram* = 3000
- *y_ram* = 16.7

Assuming that *experiment* is an instance of `Experiment`:

```
hardware_df = experiment.get_hardware_utilization()
```

get_logs()
Retrieve all log names along with their last values for this experiment.

Returns `dict` - A dictionary mapping a log names to the log's last value.

Example

Assuming that *experiment* is an instance of `Experiment`.

```
exp_logs = experiment.get_logs()
```

get_numeric_channels_values(**channel_names*)
Retrieve values of specified metrics (numeric logs).

The returned pandas.DataFrame contains 1 additional column *x* along with the requested metrics.

Parameters ***channel_names** (one or more `str`) – comma-separated metric names.

Returns

`pandas.DataFrame` - DataFrame containing values for the requested metrics.

The returned DataFrame may contain `NaN` s if one of the metrics has more values than others.

Example

Invoking `get_numeric_channels_values('loss', 'auc')` returns DataFrame with columns *x*, *loss*, *auc*.

Assuming that *experiment* is an instance of `Experiment`:

```
batch_channels = experiment.get_numeric_channels_values('batch-1-loss',
↪'batch-2-metric')
epoch_channels = experiment.get_numeric_channels_values('epoch-1-loss',
↪'epoch-2-metric')
```

Note: It's good idea to get metrics with common temporal pattern (like iteration or batch/epoch number). Thanks to this each row of returned DataFrame has metrics from the same moment in experiment. For example, combine epoch metrics to one DataFrame and batch metrics to the other.

`get_parameters``()`

Retrieve parameters for this experiment.

Returns `dict` - dictionary mapping a parameter name to value.

Examples

Assuming that *experiment* is an instance of `Experiment`.

```
exp_params = experiment.get_parameters()
```

`get_properties``()`

Retrieve User-defined properties for this experiment.

Returns `dict` - dictionary mapping a property key to value.

Examples

Assuming that *experiment* is an instance of `Experiment`.

```
exp_properties = experiment.get_properties()
```

`get_system_properties``()`

Retrieve experiment properties.

Experiment properties are for example: *owner*, *time of creation*, *time of completion*, *hostname*.

List of experiment properties may change over time.

Returns `dict` - dictionary mapping a property name to value.

Examples

Assuming that *experiment* is an instance of `Experiment`.

```
sys_properties = experiment.get_system_properties
```

`get_tags`()

Get tags associated with experiment.

Returns `list` of `str` with all tags for this experiment.

Example

Assuming that *experiment* is an instance of `Experiment`.

```
experiment.get_tags()
```

`id`

Experiment short id

Combination of project key and unique experiment number.

Format is `<project_key>-<experiment_number>`, for example: `MPI-142`.

Returns `str` - experiment short id

Examples

Assuming that *experiment* is an instance of `Experiment`.

```
exp_id = experiment.id
```

`log_artifact`(*artifact*, *destination=None*)

Save an artifact (file) in experiment storage.

Parameters

- **`artifact`** (`str`) – A path to the file in local filesystem.
- **`destination`** (`str`, optional, default is `None`) – A destination path. If `None` is passed, an artifact file name will be used.

Raises

- **`FileNotFound`** – When `artifact` file was not found.
- **`StorageLimitReached`** – When storage limit in the project has been reached.

Example

Assuming that *experiment* is an instance of `Experiment`:

```
# simple use
experiment.log_artifact('images/wrong_prediction_1.png')
```

log_graph(*graph_id*, *value*)

Upload a TensorFlow graph for this experiment.

Parameters

- **graph_id** (str) – A string UUID identifying the graph
- **value** (str) – A string representation of TensorFlow graph

Example

Assuming that:

1. *experiment* is an instance of *Experiment*,
2. *uuid* is string representation of *uuid.uuid4()*,
3. *value* is string representation of *tf.GraphDef* instance.

```
experiment.log_graph(uuid, value)
```

log_image(*log_name*, *x*, *y=None*, *image_name=None*, *description=None*, *timestamp=None*)

Log image data in Neptune

If a log with provided `log_name` does not exist, it is created automatically.
If log exists (determined by `log_name`), then new value is appended to it.
See *Limits* for information about API and storage usage upper bounds.

Parameters

- **log_name** (str) – The name of log, i.e. *mse*, *loss*, *accuracy*.
- **x** (double or PIL image) – Depending, whether y parameter is passed:
  - y not passed: The value of the log (data-point). Must be PIL image.
  - y passed: Index of log entry being appended. Must be strictly increasing.
- **y** (PIL image, optional, default is None) – The value of the log (data-point).
- **image_name** (str, optional, default is None) – Image name
- **description** (str, optional, default is None) – Image description
- **timestamp** (time, optional, default is None) – Timestamp to be associated with log entry. Must be Unix time. If None is passed, time.time() (Python 3.6 example) is invoked to obtain timestamp.

Example

Assuming that *experiment* is an instance of *Experiment*:

```
# simple use
experiment.log_image('bbox_images', PIL_object_1)
experiment.log_image('bbox_images', PIL_object_2)
experiment.log_image('bbox_images', PIL_object_3, image_name='difficult_case')
```

Note: For efficiency, logs are uploaded in batches via a queue. Hence, if you log a lot of data, you may experience slight delays in Neptune web application.

`log_metric`(*log_name*, *x*, *y=None*, *timestamp=None*)

Log metrics (numeric values) in Neptune

If a log with provided `log_name` does not exist, it is created automatically.
If log exists (determined by `log_name`), then new value is appended to it.
See *Limits* for information about API and storage usage upper bounds.

Parameters

- **`log_name`** (`str`) – The name of log, i.e. *mse*, *loss*, *accuracy*.
- **`x`** (`double`) – Depending, whether `y` parameter is passed:
  - `y` not passed: The value of the log (data-point).
  - `y` passed: Index of log entry being appended. Must be strictly increasing.
- **`y`** (`double`, optional, default is `None`) – The value of the log (data-point).
- **`timestamp`** (`time`, optional, default is `None`) – Timestamp to be associated with log entry. Must be Unix time. If `None` is passed, time.time() (Python 3.6 example) is invoked to obtain timestamp.

Example

Assuming that *experiment* is an instance of `Experiment` and 'accuracy' log does not exists:

```
# Both calls below have the same effect

# Common invocation, providing log name and value
experiment.log_metric('accuracy', 0.5)
experiment.log_metric('accuracy', 0.65)
experiment.log_metric('accuracy', 0.8)

# Providing both x and y params
experiment.log_metric('accuracy', 0, 0.5)
experiment.log_metric('accuracy', 1, 0.65)
experiment.log_metric('accuracy', 2, 0.8)
```

Note: For efficiency, logs are uploaded in batches via a queue. Hence, if you log a lot of data, you may experience slight delays in Neptune web application.

`log_text`(*log_name*, *x*, *y=None*, *timestamp=None*)

Log text data in Neptune

If a log with provided `log_name` does not exist, it is created automatically.
If log exists (determined by `log_name`), then new value is appended to it.
See *Limits* for information about API and storage usage upper bounds.

Parameters

- **`log_name`** (`str`) – The name of log, i.e. *mse*, *my_text_data*, *timing_info*.
- **`x`** (`double` or `str`) – Depending, whether `y` parameter is passed:
  - `y` not passed: The value of the log (data-point). Must be `str`.
  - `y` passed: Index of log entry being appended. Must be strictly increasing.
- **`y`** (`str`, optional, default is `None`) – The value of the log (data-point).
- **`timestamp`** (`time`, optional, default is `None`) – Timestamp to be associated with log entry. Must be Unix time. If `None` is passed, time.time() (Python 3.6 example) is invoked to obtain timestamp.

Example

Assuming that *experiment* is an instance of *`Experiment`*:

```
# common case, where log name and data are passed
neptune.log_text('my_text_data', str(data_item))

# log_name, x and timestamp are passed
neptune.log_text(log_name='logging_losses_as_text',
                 x=str(val_loss),
                 timestamp=1560430912)
```

Note: For efficiency, logs are uploaded in batches via a queue. Hence, if you log a lot of data, you may experience slight delays in Neptune web application.

`name`

Experiment name

Returns `str` experiment name

Examples

Assuming that *project* is an instance of *`Project`*.

```
experiment = project.create_experiment('exp_name')
exp_name = experiment.name
```

`remove_property`(*key*)

Removes a property with given key.

Parameters **`key`** (single `str`) – Key of property to remove.

Examples

Assuming that *experiment* is an instance of `Experiment`:

```
experiment.remove_property('host')
```

remove_tag(*tag*)
Removes single tag from the experiment.

Parameters **tag** (str) – Tag to be removed

Example

Assuming that *experiment* is an instance of `Experiment`.

```
# assuming experiment has tags: `['tag-1', 'tag-2']`.
experiment.remove_tag('tag-1')
```

Note: Removing a tag that is not assigned to this experiment is silently ignored.

reset_log(*log_name*)
Resets the log.

Removes all data from the log and enables it to be reused from scratch.

Parameters **log_name** (str) – The name of log to reset.

Raises **ChannelDoesNotExist** – When the log with name `log_name` does not exist on the server.

Example

Assuming that *experiment* is an instance of `Experiment`.

```
experiment.reset_log('my_metric')
```

Note: Check Neptune web application to see that reset charts have no data.

send_artifact(*artifact*, *destination=None*)
Save an artifact (file) in experiment storage.

Alias for `log_artifact()`

send_graph(*graph_id*, *value*)
Alias for `log_graph()`

send_image(*channel_name*, *x*, *y=None*, *name=None*, *description=None*, *timestamp=None*)
Log image data in Neptune.

Alias for `log_image()`

send_metric(*channel_name*, *x*, *y=None*, *timestamp=None*)
Log metrics (numeric values) in Neptune.

Alias for `log_metric()`

send_text (*channel_name*, *x*, *y=None*, *timestamp=None*)
Log text data in Neptune.

Alias for `log_text()`

set_property (*key*, *value*)
Set *key-value* pair as an experiment property.

If property with given `key` does not exist, it adds a new one.

Parameters

- **key** (`str`) – Property key.
- **value** (`obj`) – New value of a property.

Examples

Assuming that *experiment* is an instance of `Experiment`:

```
experiment.set_property('model', 'LightGBM')
experiment.set_property('magic-number', 7)
```

state
Current experiment state

Possible values: *'running'*, *'succeeded'*, *'failed'*, *'aborted'*.

Returns `str` - current experiment state

Examples

Assuming that *experiment* is an instance of `Experiment`.

```
state_str = experiment.state
```

stop (*exc_tb=None*)
Marks experiment as finished (succeeded or failed).

Parameters **exc_tb** (`str`, optional, default is `None`) – Additional traceback information to be stored in experiment details in case of failure (stacktrace, etc). If this argument is `None` the experiment will be marked as succeeded. Otherwise, experiment will be marked as failed.

Examples

Assuming that *experiment* is an instance of `Experiment`:

```
# Marks experiment as succeeded
experiment.stop()

# Assuming 'ex' is some exception,
# it marks experiment as failed with exception info in experiment details.
experiment.stop(str(ex))
```

3.8 Notebook

class neptune.notebook.**Notebook**(*client*, *project*, *_id*, *owner*)

Bases: object

It contains all the information about a Neptune Notebook

Parameters

- **client** (Client) – Client object
- **project** (*Project*) – Project object
- **_id** (str) – Notebook uuid
- **owner** (str) – Creator of the notebook is the Notebook owner

Examples

```
# Create a notebook in Neptune.
notebook = project.create_notebook('data_exploration.ipynb')
```

add_checkpoint(*file_path*)

Uploads new checkpoint of the notebook to Neptune

Parameters **file_path** (str) – File path containing notebook contents

Example

```
# Create a notebook.
notebook = project.create_notebook('file.ipynb')

# Change content in your notebook & save it

# Upload new checkpoint
notebook.add_checkpoint('file.ipynb')
```

get_name()

Returns the name used to upload the current checkpoint of this notebook

Returns the name of current checkpoint

Return type str

get_path()

Returns the path used to upload the current checkpoint of this notebook

Returns path of the current checkpoint

Return type str

3.9 Utils

class neptune.git_info.**GitInfo**(*commit_id*, *message=”*, *author_name=”*, *author_email=”*, *commit_date=”*, *repository_dirty=True*)

Bases: object

Class that keeps information about a git repository in experiment.

When `create_experiment()` is invoked, instance of this class is created to store information about git repository. This information is later presented in the experiment details tab in the Neptune web application.

Parameters

- **commit_id** (`str`) – commit id sha.
- **message** (`str`, optional, default is `""`) – commit message.
- **author_name** (`str`, optional, default is `""`) – commit author username.
- **author_email** (`str`, optional, default is `""`) – commit author email.
- **commit_date** (`datetime.datetime`, optional, default is `""`) – commit datetime.
- **repository_dirty** (`bool`, optional, default is `True`) – `True`, if the repository has uncommitted changes, `False` otherwise.

`neptune.utils.`**`get_git_info`**`(`*repo_path=None*`)`

Retrieve information about git repository.

If attempt fails, `None` will be returned.

Parameters **repo_path** (`str`, optional, default is `None`) –

Path to the repository from which extract information about git.
If `None` is passed, calling `get_git_info` is equivalent to calling `git.Repo(search_parent_directories=True)`. Check GitPython docs for more information.

Returns `GitInfo` - An object representing information about git repository.

Examples

```
# Get git info from the current directory
git_info = get_git_info('.')
```

3.10 Installation

Install neptune-client

```
pip install neptune-client
```

Install psutil to see hardware monitoring charts

```
pip install psutil
```

3.11 Create experiment

3.11.1 Minimal

```
import neptune

neptune.init('shared/onboarding')
neptune.create_experiment()
neptune.stop()
```

3.11.2 Basic

```
import neptune

# initialize session with Neptune
neptune.init('shared/onboarding')

# create experiment (all parameters are optional)
neptune.create_experiment(name='first-pytorch-ever',
                          params={'lr': 0.0005,
                                  'dropout': 0.2},
                          properties={'key1': 'value1',
                                      'key2': 17,
                                      'key3': 'other-value'},
                          description='write longer description here',
                          tags=['list-of', 'tags', 'goes-here', 'as-list-of-strings'],
                          upload_source_files=['training_with_pytorch.py'])

neptune.stop()
```

`params` and `properties` are standard Python dict.

3.11.3 Auto clean-up

Make use of the `with` statement to ensure that clean-up code is executed - no need to invoke `neptune.stop()`.

```
import neptune

neptune.init('shared/onboarding')

with neptune.create_experiment() as npt_exp:
    for i in range(1, 117):
        npt_exp.send_metric('iteration', i)
        npt_exp.send_metric('loss', 1 / i ** 0.5)
```

3.12 Track your work

```
# send metric (numeric value)
neptune.send_metric('log_loss', 0.753)

# send text
neptune.send_text('some-channel-name', 'evaluation time: 00:14:54')

# send image (PIL object)
neptune.send_image('image-channel-name', PIL_image)
```

(continues on next page)

(continued from previous page)

```
# send image (pass path to filse)
neptune.send_image('image-channel-name', 'path/to/image.png')

# send arbitrary artifact
neptune.send_artifact('path/to/arbitrary_data.torch')
```

3.13 Organize your work

```
# append tag
neptune.append_tag('new_tag')

# remove tag
neptune.remove_tag('remove_this_tag')

# set property
neptune.set_property('new_key', 'some_value')

# remove property
neptune.remove_property('remove_this_key')

# get experiment properties
with neptune.create_experiment() as npt_exp:
    exp_paramaters = npt_exp.get_parameters()
    print(exp_paramaters)
```

3.14 Limits

3.14.1 Storage limit

According to the pricing, storage is set per **project**:

- individual users - 5GB,
- teams - 50GB.

If you hit the limit, you can: start new project, or contact us directly at contact@neptune.ml.

3.14.2 Number of experiments limit

According to the pricing, there is such limit per **project**:

- individual users - 5k,
- teams - 50k.

If you hit the limit, you can: start new project, or contact us directly at contact@neptune.ml.

3.14.3 Experiment's logs limit

Each log type in Neptune (metric, text, image) is limited to 10k data points.

3.14.4 API calls rate limits

Neptune-client uses Python API to communicate with Neptune servers. Users are restricted to 1k requests per minute. If more requests are being placed, neptune-client will retry sending the data in the future (when usage does not approach the limit). In such case, Users may notice some delay between the actual state of the process that executes an experiment and data displayed in Neptune Web application. Extent of this effect is proportional to the number of API calls over the 1k limit.

Note: Our experiences suggests that only few AI research groups hit those limits.

CHAPTER

FOUR

BUGS, FEATURE REQUESTS AND QUESTIONS

If you find yourself in any trouble drop an issue on GitHub issues, fire a feature request on GitHub feature request or ask us on the Neptune community forum or Neptune community spectrum.

CHAPTER FIVE

INDICES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX