linode-api Documentation

Release 4.1.8b1

Linode

Mar 30, 2018

Contents:

Note: These docs are currently in development, and are therefore incomplete. For full documentation of the API, see the Official API Documentation.

This is the documentation for the official Python bindings of the Linode API V4. For API documentation, see developers.linode.com.

This library can be used to interact with all features of the Linode API, and is compatible with Python 2 and 3.

CHAPTER 1

Installation

To install through pypi:

```
pip install linode-api
```

To install from source:

```
git clone https://github.com/linode/python-linode-api
cd python-linode-api
python setup.py install
```

For more information, see our *Getting Started* guide.

CHAPTER 2

Table of Contents

2.1 Getting Started

2.1.1 Installation

The linode-api package can be installed from pypi as shown below:

```
pip install linode-api
```

If you prefer, you can clone the package from github and install it from source:

```
git clone git@github.com:Linode/python-linode-api
cd python-linode-api
python setup.py install
```

Note: This library uses the "linode" namespace. This could conflict with other libraries intended to interact with older versions of the Linode API. If you depend on a python library to interact with older versions of the Linode API, consider using a virtualenv when installing this library.

2.1.2 Authentication

In order to make requests to the Linode API, you will need a token. To generate one, log in to cloud.linode.com, and on your profile click "Create a Personal Access Token".

Note: You can also use an OAuth Token to authenticate to the API - see *OAuth* for details.

When creating a Personal Access Token, you will be prompted for what scopes the token should be created with. These scopes control what parts of your account this token may be used to access - for more information, see *OAuth Scopes*. Restricting what a token can access is more secure than creating one with access to your entire account, but can be less convenient since you would need to create a new token to access other parts of the account. For the examples on this page, your Personal Access Token must be able to view and create Linodes.

2.1.3 Listing your Linodes

Using the token you generated above, create a `LinodeClient` object that will be used for all interactions with the API.:

```
from linode import LinodeClient
client = LinodeClient(token)
```

This object will manage all requests you make through the API. Once it's set up, you can use it to retrieve and print a list of your Linodes:

```
my_linodes = client.linode.get_instances()

for current_linode in my_linodes:
    print(current_linode.label)
```

When retrieving collections of objects from the API, a list-like object is returned, and may be iterated over or indexed as a normal list.

2.1.4 Creating a Linode

In order to create a Linode, we need a few pieces of information:

- what :py:class:Region to create the Linode in
- what :py:class:Type of Linode to create
- what :py:class:Image to deploy to the new Linode.

We can query for these values similarly to how we listed our Linodes above:

```
available_regions = client.get_regions()
```

We could also use values that we know in advance to avoid the need to query the API. For example, we may know that we want a *g5-standard-4* Linode running the *linode/debian9* Image. Both objects and IDs are accepted when creating a Linode.:

```
chosen_region = available_regions[0]

new_linode, password = client.linode.create_instance(chosen_region,
                                                      'g5-standard-4',
                                                      image='linode/debian9')
```

`create_instance()` returns the newly-created Linode object and the root password that was generated for it. This Linode will boot automatically, and should be available shortly. Finally, let's print out the results so we can access our new server.

```
print("ssh root@{} - {}".format(new_linode.ipv4[0], password))
```

Continue on to Core Concepts

2.2 Core Concepts

The linode-api package, and the API V4, have a few ideas that will help you more quickly become proficient with their usage. This page assumes you've read the Getting Started guide, and know the basics of authentication already.

2.2.1 Pagination

The Linode API V4 loosely follows a RESTful design, and paginates results to responses for GETs to collections. This library handles pagination transparently, and does not load pages of data until they are required. This is handled by the `PaginatedList` class, which behaves similarly to a python list. For example:

```
linodes = client.linode.get_instances() # returns a PaginatedList of linodes

first_linode = linodes[0] # the first page is loaded automatically, this does
                          # not emit an API call

# you can also use the `first()` convenience function for this
first_linode = linodes.first()

last_linode = linodes[-1] # loads only the last page, if it hasn't been loaded yet
                          # this _will_ emit an API call if there were two or
                          # more pages of results.  If there was only one page,
                          # this does not emit an additional call

for current_linode in linodes: # iterate over all results, loading pages as necessary
    print(current_linode.label)
```

If you're not concerned about performance, using a `PaginatedList` as a normal list should be fine. If your application is sensitive to performance concerns, be aware that iterating over a `PaginatedList` can cause the thread to wait as a synchronous request for additional data is made mid-iteration.

2.2.2 Filtering

Collections of objects in the API can be filtered to make their results more useful. For example, instead of having to do this filtering yourself on the full list, you can ask the API for all Linodes you own belonging to a certain group. This library implements filtering with a SQLAlchemy-like syntax, where a model's attributes may be used in comparisons to generate filters. For example:

```
prod_linodes = client.linode.get_instances(Linode.group == "production")
```

Filters may be combined using boolean operators similar to SQLAlchemy:

```
# and_ and or_ can be imported from the linode package to combine filters
from linode import or_
prod_or_staging = client.linode.get_instances(or_(Linode.group == "production"
                                                  Linode.group == "staging"))

# and_ isn't strictly necessary, as it's the default when passing multiple
# filters to a collection
prod_and_green = client.linode.get_instances(Linode.group == "production",
                                             Linode.label.contains("green"))
```

Filters are generally only applicable for the type of model you are querying, but can be combined to your heart's content. For numeric fields, the standard numeric comparisons are accepted, and work as you'd expect. See *Filtering Collections* for full details.

2.2.3 Models

This library represents objects the API returns as "models." Most methods of *LinodeClient* return models or lists of models, and all models behave in a similar manner.

Creating Models

In addition to looking up models from collections, you can simply import the model class and create it by ID.:

```
from linode import Linode
my_linode = Linode(client, 123)
```

All models take a *LinodeClient* as their first parameter, and their ID as the second. For derived models (models that belong to another model), the parent model's ID is taken as a third argument to the constructor (i.e. to construct a `Disk` you pass a *LinodeClient*, the disk's ID, then the parent Linode's ID).

Be aware that when creating a model this way, it is _not_ loaded from the API immediately. Models in this library are **lazy-loaded**, and will not be looked up until one of their attributes that is currently unknown is accessed.

Lazy Loading

If a model is created, but not yet retrieved from the API, its attributes will be unpopulated. As soon as an unpopulated attribute is accessed, an API call is emitted to retrieve that value (and the rest of the attributes in the model) from the API. For example:

```
my_linode.id # no API call emitted - this was set on creation
my_linode.label # API call emitted - entire object is loaded from response
my_linode.group # no API call emitted - this was loaded above
```

Note: When loading a model in this fashion, if the model does not exist in the API or you do not have access to it, an ApiError is raised. If you want to load a model in a more predictable manner, see *LinodeClient.load*

Volatile Attributes

Some attributes of models are marked **volatile**. A **volatile** attribute will become stale after a short time, and if accessed when its value is stale, will refresh itself (and the entire object) from the API to ensure the value is current.:

```
my_linode.boot()
my_linode.status # booting
time.sleep(20) # wait for my_linode.status to become stale
my_linode.status # running
```

Note: While it is often safe to loop on a **volatile** attribute, be aware that there is no guarantee that their value will ever change - be sure that any such loops have another exit condition to prevent your application from hanging if something you didn't expect happens.

Updating and Deleting Models

Most models have some number of mutable attributes. Updating a model is as simple as assigning a new value to these attributes and then saving the model. Many models can also be deleted in a similar fashion.:

```
my_linode.label = "new-label"
my_linode.group = "new-group"
my_linode.save() # emits an API call to update label and group

my_linode.delete() # emits an API call to delete my_linode
```

Note: Saving a model *may* fail if the values you are attempting to save are invalid. If the values you are attemting to save are coming from an untrusted source, be sure to handle a potential `ApiError` raised by the API returning an unsuccessful response code.

Relationships

Many models are related to other models (for example a Linode has disks, configs, volumes, backups, a region, etc). Related attributes are accessed like any other attribute on the model, and will emit an API call to retrieve the related models if necessary.:

```
len(my_linode.disks) # emits an API call to retrieve related disks
my_linode.disks[0] # no API call emitted - this is already loaded

my_linode.region.id # no API call emitted - IDs are already populated
my_linode.region.country # API call emitted - retrieves region object
```

2.3 OAuth Integration

2.3.1 Overview

OAuth 2 is an open authentication protocol that describes how users can safely grant third-party applications access to some or all of their accounts with service providers. Linode implements OAuth 2 with https://login.linode.com, allowing third-party developers worlds of possibilities when integrating with Linode's service. By making an OAuth application, you can allow users to grant your app access to create, install, configure, and manage infrastructure on their behalf.

Note: If you are simply trying out the API, or if you're writing a command line tool that accepts a Personal Access Token, you can safely skip this guide.

The OAuth 2 workflow has three actors:

end user The acting user who will log in to the application.

authentication server The server that authorizes logins and issues tokens. In this case, it will be login.linode.com

client application The application you are writing, that Linode users will login to through Linode's OAuth server. You must register OAuth clients at https://cloud.linode.com or through *`create_oauth_client`* to generate a client ID and client secret (used in the exchange detailed below).

The OAuth 2 exchange works as follows:

1. The end user visits the client application's website and attempts to login using OAuth.
2. The client application redirects the end user to the authentication server with the client application's client ID and requested OAuth scopes in the query string.
3. The end user inputs their credentials to the authorization server and authorizes the login.
4. The authorization server redirects the end user to the client application with a temporary exchange code in the query string.
5. The client application issues a request to the authentication server containing the exchange code and the client application's client secret.
6. The authentication server responds to the client application with a newly issued OAuth token.

A working example of completing an OAuth exchange using this library is available in the example project Install on Linode

2.3.2 OAuth Scopes

OAuth scopes define the level of access your client application has to the accounts of users who authorize against it. While it may be easier to always request the broadest scopes, this is discouraged as it is more dangerous for the end user. The end user is presented with the requested scopes during the authorization process and may choose to abort authorization of your application based on the scopes requested.

OAuth scopes are represented by the *OAuthScopes* class, which can be used to construct lists of scopes to request. OAuth scopes are divided into "superscopes," broad categories of entities/actions that may be requested access to, and "subscopes," the level of access requested to a particular entity class. For example, if you are writing a frontend to manage NodeBalancers, you may need access to create and modify NodeBalancers, and also to list Linodes (to display more information about the individual backends). In this hypothetical case, you would likely want to construct your requested scopes like this:

```
requested_scopes = [OAuthScopes.NodeBalancer.all, OAuthScopes.Linodes.view]
```

2.3.3 Performing an OAuth Login

The *LinodeLoginClient* class manages all aspects of the OAuth exchange in this library. To create a *LinodeLoginClient*, you must use your client ID and client secret (generated upon registering a client application with Linode - see above).:

```
login_client = LinodeLoginClient(my_client_id, my_client_secret)
```

When a user attempts to login to your application using OAuth, you must issue a redirect to our authentication server (step 2 above). The *LinodeLoginClient* handles most of the details of this for you, returning the complete URL to redirect the end user to:

```
def begin_oauth_login():
    """
    An example function called when a user attempts to login user OAuth.
    """
    # generate a URL to redirect the user to, requested full access to their
    # account
    redirect_to = login_client.generate_login_url(scopes=OAuthScopes.all)

    # use your web framework to redirect the user to the generated URL
    return redirect(redirect_to)
```

Once the user has authenticated and approved this login, they will be redirected to the URL configured when your client application was registered. Your web application must accept this request, and should use it to complete the OAuth exchange (step 5 above):

```
def oauth_redirect(code=None):
    """
    An example callback function when a user authorizes this application.

    :param code: The exchange code provided by the authentication server,
                 present in the query string of the request.
    :type code: str
    """
    token, scopes = login_client.finish_oauth(code)

    # token is a valid OAuth token that may be used to construct a
    # LinodeClient and access the API on behalf of this user.
```

Now that you have been issued a token, be sure to keep it secret and specific to this user - it should be tied to their session if possible.

2.3.4 Logging Out

When a user logs out of their account, you *must* call `LoginClient.expire_token` with the token issued to your application. This will invalidate the OAuth token the user generated by logging in, which will completely revoke their session. Simply invalidating their session in your application and leaving their OAuth token active is *not* a complete logout, and should be avoided.

2.4 Linode Client

The LinodeClient is responsible for managing your connection to the API using your token. A LinodeClient is required for all connections to the API, and a reference to one is required by every model. A LinodeClient is created with a token, either an OAuth Token from the OAuth Exchange (see *oauth* for more information) or a Personal Access Token. See our *getting_started* guide for more information:

```
from linode import LinodeClient

token = "api-token" # your token goes here

client = LinodeClient(token)
```

2.4.1 Grouping

The LinodeClient class is divided into groups following the API's overall design - some methods and functions are accessible only through members of the LinodeClient class:

```
# access an ungrouped member
client.get_regions() # /regions

# access a grouped member - note the URL matches the grouping
client.linode.get_instances() # /linode/instances
```

The LinodeClient itself holds top-level collections of the API, while anything that exists under a group in the API belongs to a member of the client.

2.4.2 LinodeClient class

class `linode.`**`LinodeClient`**(*token*, *base_url='https://api.linode.com/v4'*, *user_agent=None*)

`__init__`(*token*, *base_url='https://api.linode.com/v4'*, *user_agent=None*)
The main interface to the Linode API.

Parameters

- **`token`** (*`str`*) – The authentication token to use for communication with the API. Can be either a Personal Access Token or an OAuth Token.
- **`base_url`** (*`str`*) – The base URL for API requests. Generally, you shouldn't change this.
- **`user_agent`** (*`str`*) – What to append to the User Agent of all requests made by this client. Setting this allows Linode's internal monitoring applications to track the usage of your application. Setting this is not necessary, but some applications may desire this behavior.

`account = None`
Access methods related to your account - see `AccountGroup` for more information

`create_domain`(*domain*, *master=True*, ***kwargs*)
Registers a new Domain on the acting user's account. Make sure to point your registrar to Linode's nameservers so that Linode's DNS manager will correctly serve your domain.

Parameters

- **`domain`** (*`str`*) – The domain to register to Linode's DNS manager.
- **`master`** (*`bool`*) – Whether this is a master (defaults to true)

Returns The new Domain object.

Return type Domain

`create_image`(*disk*, *label=None*, *description=None*)
Creates a new Image from a disk you own.

Parameters

- **`disk`** (*`Disk or int`*) – The Disk to imagize.
- **`label`** (*`str`*) – The label for the resulting Image (defaults to the disk's label.
- **`description`** (*`str`*) – The description for the new Image.

Returns The new Image.

Return type Image

`create_nodebalancer`(*region*, ***kwargs*)
Creates a new NodeBalancer in the given Region.

Parameters **`region`** (*`Region or str`*) – The Region in which to create the NodeBalancer.

Returns The new NodeBalancer

Return type NodeBalancer

`create_volume`(*label*, *region=None*, *linode=None*, *size=20*, ***kwargs*)
Creates a new Block Storage Volume, either in the given Region or attached to the given Linode.

Parameters

- **label** (*str*) – The label for the new Volume.
- **region** (*Region or str*) – The Region to create this Volume in. Not required if *linode* is provided.
- **linode** (*Linode or int*) – The Linode to attach this Volume to. If not given, the new Volume will not be attached to anything.
- **size** (*int*) – The size, in GB, of the new Volume. Defaults to 20.

Returns The new Volume.

Return type Volume

get_account()

Retrieves information about the acting user's account, such as billing information.

Returns Returns the acting user's account information.

Return type *Account*

get_domains(**filters*)

Retrieves all of the Domains the acting user has access to.

Parameters **filters** – Any number of filters to apply to this query.

Returns A list of Domains the acting user can access.

Return type PaginatedList of Domain

get_images(**filters*)

Retrieves a list of available Images, including public and private Images available to the acting user. You can filter this query to retrieve only Images relevant to a specific query, for example:

```
debian_images = client.get_images(
    Image.vendor == "debain")
```

Parameters **filters** – Any number of filters to apply to the query.

Returns A list of available Images.

Return type PaginatedList of Image

get_nodebalancers(**filters*)

Retrieves all of the NodeBalancers the acting user has access to.

Parameters **filters** – Any number of filters to apply to this query.

Returns A list of NodeBalancers the acting user can access.

Return type PaginatedList of NodeBalancers

get_profile()

Retrieve the acting user's Profile, containing information about the current user such as their email address, username, and uid.

Returns The acting user's profile.

Return type Profile

get_regions(**filters*)

Returns the available Regions for Linode products.

Parameters **filters** – Any number of filters to apply to the query.

Returns A list of available Regions.

Return type PaginatedList of Region

get_volumes(**filters*)
Retrieves the Block Storage Volumes your user has access to.

Parameters **filters** – Any number of filters to apply to this query.

Returns A list of Volumes the acting user can access.

Return type PaginatedList of Volume

linode = None
Access methods related to Linodes - see *LinodeGroup* for more information

load(*target_type*, *target_id*, *target_parent_id=None*)
Constructs and immediately loads the object, circumventing the lazy-loading scheme by immediately making an API request. Does not load related objects.

For example, if you wanted to load a `Linode` object with ID 123, you could do this:

```
loaded_linode = client.load(Linode, 123)
```

Similarly, if you instead wanted to load a `NodeBalancerConfig`, you could do so like this:

```
loaded_nodebalancer_config = client.load(NodeBalancerConfig, 456, 432)
```

Parameters

- **target_type** (*type*) – The type of object to create.
- **target_id** (*int or str*) – The ID of the object to create.
- **target_parent_id** (*int, str, or None*) – The parent ID of the object to create, if applicable.

Returns The resulting object, fully loaded.

Return type target_type

Raises **ApiError** – if the requested object could not be loaded.

longview = None
Access information related to the Longview service - see *LongviewGroup* for more inforamtion

networking = None
Access methods related to networking on your account - see *NetworkingGroup* for more information

profile = None
Access methods related to your user - see *ProfileGroup* for more information

support = None
Access methods related to support - see *SupportGroup* for more information

2.4.3 Groups

These groups are accessed off of the *LinodeClient* class by name. For example:

```
client.linode.get_instances()
```

See *LinodeClient* for more information on the naming of these groups, although generally they are named the same as the first word of the group.

LinodeGroup

Includes methods for managing and creating Linodes, as well as accessing and working with associated features.

class linode.linode_client.**LinodeGroup**(*client*)

Encapsulates Linode-related methods of the *LinodeClient*. This should not be instantiated on its own, but should instead be used through an instance of *LinodeClient*:

```
client = LinodeClient(token)
linodes = client.linode.get_instances() # use the LinodeGroup
```

This group contains all features beneath the */linode* group in the API v4.

create_instance(*ltype*, *region*, *image=None*, *authorized_keys=None*, ***kwargs*)

Creates a new Linode. This function has several modes of operation:

Create a Linode from an Image

To create a Linode from an `Image`, call *create_instance* with a `Type`, a `Region`, and an `Image`. All three of these fields may be provided as either the ID or the appropriate object. In this mode, a root password will be generated and returned with the new Linode object. For example:

```
new_linode, password = client.linode.create_instance(
    "g5-standard-1",
    "us-east",
    image="linode/debian9")

ltype = client.linode.get_types().first()
region = client.get_regions().first()
image = client.get_images().first()

another_linode, password = client.linode.create_instance(
    ltype,
    region,
    image=image)
```

Create a Linode from StackScript

When creating a Linode from a `StackScript`, an `Image` that the StackScript support must be provided.. You must also provide any required StackScript data for the script's User Defined Fields.. For example, if deploying StackScript 10079 (which deploys a new Linode with a user created from keys on github:

```
stackscript = StackScript(client, 10079)

new_linode, password = client.linode.create_instance(
   "g5-standard-2",
   "us-east",
   image="linode/debian9",
   stackscript=stackscript,
   stackscript_data={"gh_username": "example"})
```

In the above example, "gh_username" is the name of a User Defined Field in the chosen StackScript. For more information on StackScripts, see the StackScript guide.

Create a Linode from a Backup

To create a new Linode by restoring a `Backup` to it, provide a `Type`, a `Region`, and the `Backup` to restore. You may provide either IDs or objects for all of these fields:

```
existing_linode = Linode(client, 123)
snapshot = existing_linode.available_backups.snapshot.current

new_linode = client.linode.create_instance(
    "g5-standard-1",
    "us-east",
    backup=snapshot)
```

Create an empty Linode

If you want to create an empty Linode that you will configure manually, simply call *create_instance* with a `Type` and a `Region`:

```
empty_linode = client.linode.create_instance("g5-standard-2", "us-east")
```

When created this way, the Linode will not be booted and cannot boot successfully until disks and configs are created, or it is otherwise configured.

Parameters

- **ltype** (*str or LinodeType*) – The Linode Type we are creating
- **region** (*str or Region*) – The Region in which we are creating the Linode
- **image** (*str or Image*) – The Image to deploy to this Linode. If this is provided and no root_pass is given, a password will be generated and returned along with the new Linode.
- **stackscript** (*int or StackScript*) – The StackScript to deploy to the new Linode. If provided, "image" is required and must be compatible with the chosen StackScript.
- **stackscript_data** (*dict*) – Values for the User Defined Fields defined in the chosen StackScript. Does nothing if StackScript is not provided.
- **backup** (*int of Backup*) – The Backup to restore to the new Linode. May not be provided if "image" is given.
- **authorized_keys** (*list or str*) – The ssh public keys to install in the linode's /root/.ssh/authorized_keys file. Each entry may be a single key, or a path to a file containing the key.
- **label** (*str*) – The display label for the new Linode
- **group** (*str*) – The display group for the new Linode
- **booted** (*bool*) – Whether the new Linode should be booted. This will default to True if the Linode is deployed from an Image or Backup.

Returns A new Linode object, or a tuple containing the new Linode and the generated password.

Return type Linode or tuple(Linode, str)

Raises

- **ApiError** – If contacting the API fails
- **UnexpectedResponseError** – If the API resposne is somehow malformed. This usually indicates that you are using an outdated library.

create_stackscript (*label*, *script*, *images*, *desc=None*, *public=False*, ***kwargs*)

Creates a new `StackScript` on your account.

Parameters

- **label** (*str*) – The label for this StackScript.
- **script** (*str*) – The script to run when a `Linode` is deployed with this StackScript. Must begin with a shebang (#!).
- **images** (*list of Image*) – A list of `Images` that this StackScript supports. Linodes will not be deployed from this StackScript unless deployed from one of these Images.
- **desc** (*str*) – A description for this StackScript.
- **public** (*bool*) – Whether this StackScript is public. Defaults to False. Once a StackScript is made public, it may not be set back to private.

Returns The new StackScript

Return type StackScript

get_instances(**filters*)

Returns a list of Linodes on your account. You may filter this query to return only Linodes that match specific criteria:

```
prod_linodes = client.linode.get_instances(Linode.group == "prod")
```

Parameters **filters** – Any number of filters to apply to this query.

Returns A list of Linodes that matched the query.

Return type PaginatedList of Linode

get_kernels(**filters*)

Returns a list of available `Kernels`. Kernels are used when creating or updating `LinodeConfigs, LinodeConfig>`.

Parameters **filters** – Any number of filters to apply to this query.

Returns A list of available kernels that match the query.

Return type PaginatedList of Kernel

get_stackscripts(**filters*, ***kwargs*)

Returns a list of `StackScripts`, both public and private. You may filter this query to return only `StackScripts` that match certain criteria. You may also request only your own private `StackScripts`:

```
my_stackscripts = client.linode.get_stackscripts(mine_only=True)
```

Parameters

- **filters** – Any number of filters to apply to this query.
- **mine_only** (*bool*) – If True, returns only private StackScripts

Returns A list of StackScripts matching the query.

Return type PaginatedList of StackScript

get_types(**filters*)

Returns a list of Linode types. These may be used to create or resize Linodes, or simply referenced on their own. Types can be filtered to return specific types, for example:

```
standard_types = client.linode.get_types(Type.class == "standard")
```

Parameters **`filters`** – Any number of filters to apply to the query.

Returns A list of types that match the query.

Return type PaginatedList of Type

AccountGroup

Includes methods for managing your account.

class `linode.linode_client.`**`AccountGroup`**(*client*)

`create_oauth_client`(*name*, *redirect_uri*, ***kwargs*)
Make a new OAuth Client and return it

`get_invoices`()
Returns Invoices issued to this account

`get_oauth_clients`(**filters*)
Returns the OAuth Clients associated to this account

`get_payments`()
Returns a list of Payments made to this account

`get_settings`()
Resturns the account settings data for this acocunt. This is not a listing endpoint.

`get_transfer`()
Returns a MappedObject containing the account's transfer pool data

`get_users`(**filters*)
Returns a list of users on this account

`mark_last_seen_event`(*event*)
Marks event as the last event we have seen. If event is an int, it is treated as an event_id, otherwise it should be an event object whose id will be used.

ProfileGroup

Includes methods for managing your user.

class `linode.linode_client.`**`ProfileGroup`**(*client*)
Collections related to your user.

`create_personal_access_token`(*label=None*, *expiry=None*, *scopes=None*, ***kwargs*)
Creates and returns a new Personal Access Token

`get_apps`(**filters*)
Returns the Authorized Applications for this user

`get_tokens`(**filters*)
Returns the Person Access Tokens active for this user

NetworkingGroup

Includes methods for managing your networking systems.

class `linode.linode_client.`**`NetworkingGroup`**(*client*)

allocate_ip(*linode*)

Allocates an IP to a Linode you own. Additional IPs must be requested by opening a support ticket first.

Parameters **linode** (*Linode or int*) – The Linode to allocate the new IP for.

Returns The new IPAddress

Return type IPAddress

assign_ips(*region*, **assignments*)

Redistributes `IP Addressees` within a single region. This function takes a `Region` and a list of assignments to make, then requests that the assignments take place. If any `Linode` ends up without a public IP, or with more than one private IP, all of the assignments will fail.

Example usage:

```
linode1 = Linode(client, 123)
linode2 = Linode(client, 456)

# swap IPs between linodes 1 and 2
client.networking.assign_ips(linode1.region,
                             linode1.ips.ipv4.public[0].to(linode2),
                             linode2.ips.ipv4.public[0].to(linode1))
```

Parameters

- **region** (*str or Region*) – The Region in which the assignments should take place. All Linodes and IPAddresses involved in the assignment must be within this region.
- **assignments** (*dct*) – Any number of assignments to make. See `IPAddress.to` for details on how to construct assignments.

LongviewGroup

Includes methods for interacting with our Longview service.

class linode.linode_client.**LongviewGroup**(*client*)

create_client(*label=None*)

Creates a new LongviewClient, optionally with a given label.

Parameters **label** – The label for the new client. If None, a default label based on the new client's ID will be used.

Returns A new LongviewClient

Raises

- **ApiError** – If a non-200 status code is returned
- **UnexpectedResponseError** – If the returned data from the api does not look as expected.

get_clients(**filters*)

Requests and returns a paginated list of LongviewClients on your account.

get_subscriptions(**filters*)

Requests and returns a paginated list of LongviewSubscriptions available

SupportGroup

Includes methods for viewing and opening tickets with our support department.

class linode.linode_client.**SupportGroup**(*client*)

open_ticket(*summary*, *description*, *regarding=None*)

2.5 Linode Login Client

The *LinodeLoginClient* is the primary interface to the login.linode.com OAuth service, and only needs to be used if writing an OAuth application. For an example OAuth application, see Install on Linode, and for a more comprehensive overview of OAuth, read our *OAuth guide*.

2.5.1 LinodeLoginClient class

Your interface to Linode's OAuth authentication server.

class linode.**LinodeLoginClient**(*client_id*, *client_secret*, *base_url='https://login.linode.com'*)

__init__(*client_id*, *client_secret*, *base_url='https://login.linode.com'*)

Create a new LinodeLoginClient. These clients do not make any requests on creation, and can safely be created and thrown away as needed.

For complete usage information, see the *OAuth guide*.

Parameters

- **client_id** (*str*) – The OAuth Client ID for this client.
- **client_secret** (*str*) – The OAuth Client Secret for this client.
- **base_url** (*str*) – The URL for Linode's OAuth server. This should not be changed.

expire_token(*token*)

Given a token, makes a request to the authentication server to expire it immediately. This is considered a responsible way to log out a user. If you simply remove the session your application has for the user without expiring their token, the user is not _really_ logged out.

Parameters **token** (*str*) – The OAuth token you wish to expire

Returns If the expiration attempt succeeded.

Return type bool

Raises **ApiError** – If the expiration attempt failed.

finish_oauth(*code*)

Given an OAuth Exchange Code, completes the OAuth exchange with the authentication server. This should be called once the user has already been directed to the login_uri, and has been sent back after successfully authenticating. For example, in Flask, this might be implemented as a route like this:

```
@app.route("/oauth-redirect")
def oauth_redirect():
    exchange_code = request.args.get("code")
    login_client = LinodeLoginClient(client_id, client_secret)
    token, scopes = login_client.finish_oauth(exchange_code)

    # store the user's OAuth token in their session for later use
    # and mark that they are logged in.

    return redirect("/")
```

Parameters **`code`** (*`str`*) – The OAuth Exchange Code returned from the authentication server in the query string.

Returns The new OAuth token, and a list of scopes the token has.

Return type tuple(str, list)

Raises **`ApiError`** – If the OAuth exchange fails.

`generate_login_url`(*scopes=None*, *redirect_uri=None*)

Generates a url to send users so that they may authenticate to this application. This url is suitable for redirecting a user to. For example, in Flask, a login route might be implemented like this:

```
@app.route("/login")
def begin_oauth_login():
    login_client = LinodeLoginClient(client_id, client_secret)
    return redirect(login_client.generate_login_url())
```

Parameters

- **`scopes`** (*`list`*) – The OAuth scopes to request for this login.
- **`redirect_uri`** (*`str`*) – The requested redirect uri. The login service enforces that this is under the registered redirect path.

Returns The uri to send users to for this login attempt.

Return type str

2.5.2 OAuth Scopes

When requesting authorization to a user's account, OAuth Scopes allow you to specify the level of access you are requesting.

class `linode.login_client.`**`OAuthScopes`**

Represents the OAuth Scopes available to an application. In general, an application should request no more scopes than it requires. This class should be treated like a Enum, and used as follows:

```
required_scopes = [OAuthScopes.Linodes.all, OAuthScopes.Domains.read_only]
```

Lists of OAuth Scopes are accepted when calling the *`generate_login_url`* method of the *`LinodeLoginClient`*.

All contained enumerations of OAuth Scopes have two levels, "read_only" and "read_write". "read_only" access grants you the ability to get resources and of that type, but not to change, create, or delete them. "read_write" access allows to full access to resources of the requested type. In the above example, you are requesting access to view, modify, create, and delete Linodes, and to view Domains.

class Account

Access to the user's account, including billing information, tokens management, user management, etc.

class Clients
An enumeration.

class Domains
Access to Domains

class Events
Access to a user's Events

class IPs
Access to IPs and networking managements

class Linodes
Access to Linodes

class NodeBalancers
Access to NodeBalancers

class StackScripts
Access to private StackScripts

class Tickets
Access to view, open, and respond to Support Tickets

class Tokens
An enumeration.

class Users
An enumeration.

class Volumes
Access to Block Storage Volumes

all = *
If necessary, an application may request all scopes by using OAuthScopes.all

2.6 Pagination

The Linode API V4 returns collections of resources one page at a time. While this is useful, this library abstracts away the details of pagination and makes collections of resources appear as a single, uniform list that can be accessed, iterated over, and indexed as any normal Python list would be:

```
regions = client.get_regions() # get a collection of Regions

for region in regions:
    print(region.id)

first_region = regions[0]
last_region = regions[-1]
```

Pagination is handled transparently, and as requested. For example, if you had three pages of Linodes, accessing your collection of Linodes would behave like this:

```
linodes = client.linode.get_instances() # loads the first page only

linodes[0] # no additional data is loaded

linodes[-1] # third page is loaded to retrieve the last Linode in the collection
```

```
for linode in linodes:
    # the second page will be loaded as soon as the first Linode on that page
    # is required.  The first and third pages are already loaded, and will not
    # be loaded again.
    print(linode.label)
```

The first page of a collection is always loaded when the collection is returned, and subsequent pages are loaded as they are required. When slicing a paginated list, only the pages required for the slice are loaded.

2.6.1 PaginatedList class

class linode.**PaginatedList**(*client*, *page_endpoint*, *page=[]*, *max_pages=1*, *total_items=None*, *parent_id=None*, *filters=None*)

The PaginatedList encapsulates the API V4's pagination in an easily consumable way. A PaginatedList may be treated like a normal *list* in all ways, and can be iterated over, indexed, and sliced.

PaginatedLists should never be constructed manually, and instead should be created by requesting a collection of resources from the `LinodeClient`. For example:

```
linodes = client.linode.get_instances() # returns a PaginatedList of Linodes
```

Once you have a PaginatedList of resources, it doesn't matter how many resources the API will return - you can iterate over all of them without having to worry about pagination.:

```
# iterate over all linodes.  If there are two or more pages,
# they will be loaded as required.
for linode in linodes:
    print(linode.label)
```

You may access the number of items in a collection by calling *len* on the PaginatedList:

```
num_linodes = len(linodes)
```

This will _not_ emit another API request.

first()

A convenience method for getting only the first item in this list. Exactly equivalent to getting index 0.

Returns The first item in this list.

last()

A convenience method for getting only the last item in this list. Exactly equivalent to getting index -1.

Returns The first item in this list.

only()

Returns the first item in this list, and asserts that it is the only item. This is useful when querying a collection for a resource and expecting to get only one back. For instance:

```
# raises if it finds more than one Linode
production_box = client.linode.get_instances(Linode.group == "prod").only()
```

Returns The first and only item in this list.

Raises **ValueError** – If more than one item is in this list.

2.7 Filtering Collections

Collections returned by the *LinodeClient* can be filtered using a SQLAlchemy-like syntax. When calling any "get" method of the *LinodeClient* class of one of its groups, any number of filters may be passed in as boolean comparisons between attributes of the model returned by the collection.

For example, calling *get_instances* returns a list of `Linode` objects, so we can use properties of `Linode` to filter the results:

```
# returns all Linodes in the "prod" group
client.linode.get_instances(Linode.group == "prod")
```

You can use any boolean comparisons when filtering collections:

```
# returns all Linodes _not_ in us-east-1a
client.linode.get_instances(Linode.region != "us-east-1a")
```

You can combine filters to be even more specific - by default all filters are considered:

```
# returns all Linodes in the "prod" group that are in us-east-1a
client.linode.get_instances(Linode.group == "prod",
                            Linode.region == "us-east-1a")
```

If you need to combine the results of two filters, you can use *or_* to define this relationship:

```
# returns all Linodes in either the "prod" or "staging" groups
client.linode.get_instances(or_(Linode.group == "prod",
                                Linode.group == "staging"))
```

*and_* is also available in case you need to do deeply-nested comparisons:

```
# returns all Linodes in the group "staging" and any Linodes in the "prod"
# group that are located in "us-east-1a"
client.linode.get_instances(or_(Linode.group == "staging",
                                and_(Linode.group == "prod",
                                     Linode.region == "us-east-1a"))
```

class `linode.objects.filtering.`**`Filter`**(*dct*)

A Filter represents a comparison to send to the API. These should not be constructed normally, but instead should be returned from comparisons between class attributes of filterable classes (see above). Filters can be combined with *and_* and *or_*.

`linode.objects.filtering.`**`and_`**(*a*, *b*)

Combines two *Filters* with an "and" operation, matching any results that match both of the given filters.

Parameters

- **a** (`Filter`) – The first filter to consider.
- **b** (`Filter`) – The second filter to consider.

Returns A filter that matches both a and b

Return type *Filter*

`linode.objects.filtering.`**`limit`**(*amount*)

Allows limiting of results in a collection. You may only ever apply a limit once per request. For example:

```
# returns my first 5 Linodes
client.linode.get_instances(limit(5))
```

Parameters **amount** (*int*) – The number of results to return.

Returns A filter that will limit the number of results returned.

Return type *Filter*

`linode.objects.filtering.`**`or_`**(*a*, *b*)

Combines two *Filters* with an "or" operation, matching any results that match any of the given filters.

Parameters

- **a** (Filter) – The first filter to consider.
- **b** (Filter) – The second filter to consider.

Returns A filter that matches either a or b

Return type *Filter*

`linode.objects.filtering.`**`order_by`**(*field*, *desc=False*)

Allows ordering of results. You may only ever order a collection's results once in a given request. For example:

```
# sort results by Linode group
client.linode.get_instances(order_by(Linode.group))
```

Parameters

- **field** (*FilterableAttribute*) – The field to order results by. Must be a filterable attribute of the model.
- **desc** (*bool*) – If True, return results in descending order. Defaults to False

Returns A filter that will order results as requested.

Return type *Filter*

Python Module Index

Index