DOPYAPI

Release 0.1.0

Oct 31, 2020

User Guide

CHAPTER 1

Installation

To install the library use this command:

```
pip3 install git+git://github.com/mohsenSy/dopyapi.git
```

CHAPTER 2

Getting Started

We will explain here the basic usage of **dopyapi**, using simple code.

2.1 Get an access token

Before you start using the library you must acquire an access token from your Digital Ocean account, to get one visit this URL, click on `Generate New Token` then save the token you see somewhere safe and make sure to keep it because you will not be able to see it again.

Export the `DO_TOKEN` variable in your shell session to be able to use it later in the code:

```
export DO_TOKEN=<access token>
```

2.2 Print Account Information

The following code prints user information in JSON format, notice we did not pass any value to `do.authenticate()` because it will use the value stored in *DO_TOKEN* when we do not pass any value to it:

```
import dopyapi as do
do.authenticate()
account = do.Account()
print(account.json())
```

Every class for a Digital Ocean has the `json` method to return a dictionary that contains all attributes for that resource with their values.

You can print account information individually like this:

```
print(account.email) # Print user email
print(account.droplet_limit) # Print how many droplets can a user create
```

2.3 List all available droplets

Now let us try to list all available droplets:

```
droplets = do.Droplet.list()
for droplet in droplets:
  print(droplet)
```

Each resource in Digital Ocean is represented by a class whose name starts with a capital letter and is singular, and all these classes has the `list` method that returns a list of objects for this resource, this method takes these two shared parameters:

- page: The number of page to fetch resources from it, default is 1
- per_page: The number of resources returned, default is 20

2.4 Create a new droplet

To create a new droplet use this code:

```
droplet = do.Droplet()
droplet_data = {
  "name": "droplet1",
  "image": do.images.ubuntu,
  "size": do.sizes.tiny,
  "region": "ams3"
}
droplet.create(**droplet_data)
print(droplet.getPublicIP())
```

In this code we created a new object of *`Droplet`*, then we prepared a dictionary to hold the values of attributes that will be used when creating the droplet, then we call the method *`create()`*, this method is used by all classes to create new resources.

Lastly we use the method *`getPublicIP()`* to print the public IP address of the new droplet, this method will wait untill the droplet is ready.

2.5 Take snapshot of a Droplet

In Digital Ocean API, taking a Droplet snapshot is considered as an action, and here in our library we have methods for each action associated with a resource that has the same action's name and takes named parameters, and returns an action object, the following code takes a snapshot of a droplet and uses the returned action object to wait until the snapshot is finished:

```
droplets = do.Droplet.list()
droplet = droplets[0]
action = droplet.snapshot(name="s1")
while action.status == "in-progress":
  action.load()
if action.status == "completed":
  print("snapshot was finished successfully")
else:
  print("There was an error with snapshot")
```

In the previous code we fetch a list of droplets first, then we take the first droplet from the list and call the method `snapshot` on it.

Then we use a while loop and check the status of the action until it is not in-progress anymore, after that if the status was "completed" we print a success message and if it was not "completed" we print an error message.

Note: Notice that the `snapshot` method does not have a link to a method in the class *`Droplet`*, that is because we use python's magic methods to implement actions and many more features here, you can learn more about our use of magic methods in *Magic Methods*

2.6 List Droplet snapshots

To list the snapshots of a droplet use this code:

```
snapshots = droplet.listSnapshots()
for snapshot in snapshots:
  print(snapshot)
```

Here every element of the list is an instance of *`Snapshot`*.

2.7 Create a new firewall and assign it to a droplet

To create a firewall we need to prepare objects of *`Location`*, *`InboundRule`* and *`OutboundRule`*.

The following code shows how to do it:

```
location_local = do.Location(addresses=["192.168.2.0/24"]) # This defines a location␣
↪that matches all IP addresses in subnet "192.168.2.0/24"
location_all = do.Location(addresses=["0.0.0.0/0"])
inbound_rule = do.InboundRule(ports="1234", protocol="udp", sources=location_local) #␣
↪this defines an inbound rule for protocol udp and port 1234 using the previous␣
↪location as source of traffic
outbound_rule = do.OutboundRule(destinations=location_all) # This defines an outbound␣
↪rule that matches all tcp traffic to all ports and destinations
firewall = do.Firewall()
firewall.create(name="fw1", inbound_rules=[inbound_rule], outbound_rules=[outbound_
↪rule])
droplets = do.Droplet.list()
firewall.addDroplets(droplets[0])
```

In the previous code we used two objects of *`Location`*, to hold the addresses of a local subnet and also all available addresses, the first one is used to allow traffic from local network and the other to allow traffic to all addresses.

We also used two objects of *`InboundRule`* and *`OutboundRule`* to add two rules to our firewall, then we declared an object of class *`Firewall`* and used the *`create()`* method to create the firewall.

Then we retrieved a list of Digital Ocean droplets and assigned the firewall to the first droplet.

2.8 List and create SSH keys

In order to list all available SSH keys you need to use the *list()* method just like all other resources that has this method available:

```
ssh_keys = do.SSHKey.list()
for ssh_key in ssh_keys:
  print(ssh_key)
```

To create an SSH key we need to specify its name and public key as follows:

```
import os
ssh_key = do.SSHKey()
public_key = open(f"{os.environ['HOME']}/.ssh/id_rsa.pub", "r").read()
ssh_key_data = {
  "name": "new-key",
  "public_key": public_key
}
ssh_key.create(**ssh_key_data)
```

You need to have a public key available at `~/.ssh/id_rsa.pub` and also this public must not exist in your account or you will get *ClientError* with this message `SSH Key is already in use on your account.`

2.9 List images

Use the following code to list available images:

```
images = do.Image.list(page=2, per_page=30)
for image in images:
  print(image)
user_images = do.Image.listUser()
for image in user_images:
  print(image)
app_images = do.Image.listApplication()
for image in app_images:
  print(image)
```

The previous code uses the `page` and `per_page` parameters to choose the page of images to fetch from the API, here we are fetching the second 30 images instead of the first 20 by default, we also used *listUser()* to list user private images and *listApplication()* to list application images available in the market place.

CHAPTER 3

Magic Methods

Note: This part is intended to describe an internal aspect of the library that is only needed by people who want to develop the library itself and not use it if you only want to use the library in your own projects then you may skip this part however if you have some good python experience and want to know the library internally then go ahead and read.

3.1 What are python magic methods

Magic methods are like normal methods in Python but they are not called directly by programmers, they are called by python interpreter behind the scenes each one of them at specific time or in a specific situation, for example we have `__init__` magic method that is called when we create an object of a class, `__del__` which is called when we delete an object of a class or it gets out of scope.

These methods start with __ and end with it too, so when we name methods we better not use this syntax.

Note: We will not explain all python magic methods here, to learn more follow this link.

3.2 Attribute get and set magic methods

There are two magic methods used here the first one is `__getattribute__` which is called every time we access an attribute of an object, and `__setattr__` that is called every time we set a value for an attribute.

Look at this code:

```
class A:
  def __getattribute__(self, attr):
    try:
```

(continues on next page)

(continued from previous page)

```
      return object.__getattribute__(self, attr)
    except AttributeError:
      return "attribute not found"
  def __setattr__(self, attr, value):
    object.__setattr__(self, attr, value)
a = A()
a.x = 3
print(a.x)
print(a.y)
```

In the previous code we used these methods to explain their usage. Every time we try to set a value for an attribute, as found in line 10 `a.x = 3`, the method `__setattr__` is called, which calls the `__setattr__` method on the object called `object` that is the parent for all objects in python.

And every time we try to access an attribute value, the method `__getattribute__` is called and tries to get the value using the `object` object, if it is not available then it returns the string "attribute not found", this is for demonstration purposes only.

3.3 Resource class magic methods

In our library we have the class *`Resource`*, that is the parent for all Digital Ocean resources, most of the magic happens in this class that was written carefully to help us implement other functionality easily, this class has these two magic methods `__getattribute__()` and `__setattr__()`.

Here is the code for the `__getattribute__` method:

```
if attr == "resource":
      return object.__getattribute__(self, attr)
  resource = object.__getattribute__(self, "resource")
  static_attrs = resource.static_attrs
  dynamic_attrs = resource.dynamic_attrs
  fetch_attrs = resource.fetch_attrs
  action_attrs = resource.action_attrs
  if attr in static_attrs or attr in dynamic_attrs or attr in fetch_attrs:
      return self.__fetch(attr)
  if attr in action_attrs:
      return lambda **kwargs : self.action(type=attr, **kwargs)
  return object.__getattribute__(self, attr)
```

From the previous code we can see that the *resource* attribute is treated first, this attribute holds the class of Digital Ocean resource, if we are using *`Droplet`* then instance would be `Droplet`, this class has a list of class attributes that define the resource, some of them are `static_attrs`, `dynamic_attrs`, `fetch_attrs` and `action_attrs`, we will explain them now.

- `static_attrs`: These attributes are set by Digital Ocean and cannot be changed directly.
- `dynamic_attrs`: These attributes are set by users and can be changed, they are used when creating resources or updating them.
- `fetch_attrs`: These attributes are set by Digital Ocean and can be used to fetch resources based on them for example: we can fetch a droplet based on its ID.
- `action_attrs`: These are the defined actions for the resource, if a resource does not have any actions associated with it then this list will be empty.

If you check the code again, you can see that if the attribute is one of `fetch_attrs`, `static_attrs` or `dynamic_attrs` we call `__fetch` method, whcih checks if we previously fetched from the API and populated the

object with data, it simply returns the value for the attribute, if not it calls __do_fetch that will fetch data from the API.

If the attribute is in action_attrs, it returns a lambda function that calls the *action()* method with the correct type.

With this method we transparently call API only when needed, but what if the value of one attribute changes? How can we detect this and refresh from the API? The answer is in __setattr__ magic method.

Here is the code for the __setattr__ magic method:

```
if attr == "resource":
      object.__setattr__(self, attr, value)
  resource = object.__getattribute__(self, "resource")
  static_attrs = resource.static_attrs
  dynamic_attrs = resource.dynamic_attrs
  fetch_attrs = resource.fetch_attrs
  if attr in fetch_attrs:
      self.__dict__["__changed"] = attr
      self.__dict__["__fetched"] = False
  if attr in static_attrs:
      return
  self.__dict__[attr] = value
```

From the previous code we can see, if the attribute is in fetch_attrs then we set the value for __changed to the name of the attribute and __fetched to False, with this way if we try to get the value of an attribute the class can detect the change and call the API to update.

We can also see that if the attribute is in static_attrs it returns without updating its value because these attributes are set by Digital Ocean and cannot be changed directly.

With these two methods we gave our library the ability to call the API transparently when needed and help users use our classes as they would do with any other classes.

CHAPTER 4

How to use the Library?

In this section we will describe the structure of the library needed by the users who want to consume it.

4.1 Digital Ocean resources as classes

Every resource in Digital Ocean is represented by a class here, this class inherits from *Resource* class, you do not need to use the *Resource* class directly just use the classes that inherit from it.

The class that represents a resource has a name that is singular, and starts with a capital letter for the resource, for example: droplets have a class called `Droplet`, Firewalls have a class called `Firewall` and so on.

4.2 Common methods

Every class that represents a Digital Ocean resource has these common methods

- *list()*: This method is used to retrieve instances of the resource based on its parameters which include `page` and `per_page` which default to 1 and 20 respectively, these define how many instances are returned and where to start returning them, for example: `page = 3` and `per_page = 40` means return 40 instances starting from page 3 if we had a total of 120 instances for this resource then we will return the last 40.
- *create()*: This method is used to create instances of resources, you pass the names of dynamic attributes along with their values as parameters and it creates the resource then updates the current object.
- *save()*: This method is used to save any changes we do to the dynamic attributes of an object, it sends a PUT request along with the values of all dynamic attributes to save them all.
- *json()*: This method returns a json representation of the resource, that is a python dictionary with one key equals to the resource name and its value is another dictionary which contains values for all attributes.
- *delete()*: The delete method is used to delete an instance of a resource.
- *listActions()*: This method is used to list all actions associated with a resource.
- *getAction()*: This method is used to fetch the action associated with a resource using its action ID.

- *getID()*: This method returns the value for the ID attribute, it could be a string or integer according to the resource.

4.3 Fetch single instance of a resource

Now we will talk about an important part of our Library, how to fetch resources based on the value of an attribute?

From Digital Ocean documentation it says that we can for example fetch a droplet based on its ID, so how to do this here?

If you are thinking about a method such as `get_by_id` then you are wrong, we do not use methods here to do the job, we simply assign a value to the ID attribute and we have the object ready to get values for any attribute we want, check this code to understand better:

```
import dopyapi as do
do.authenticate()
droplets = do.Droplet.list() # We fetched a list of all droplets
print(droplets[0]) # Here we print the droplet object, which will show its ID
droplet = do.Droplet() # we declare an object of class Droplet
droplet.id = droplets[0].id # Here we just assign a value for the ID attribute
print(droplet.json()) # Here we print a JSON representation of the droplet
```

You will notice that the ID of the last object `droplet` is the same as the ID of the first object of the list `droplets` and also all of its attribute values are the same too.

Where is the API call? It is actually hidden inside the *Resource* this call is only made once when we need it, so we do not bother our selves with the call.

All other classes use the same technique, each resource has its own attributes that can be used when fetching for example: Volumes can be fetched by ID, Floating IPs can be fetched by their IP address, Images can be fetched by their ID or slug. etc...

CHAPTER 5

Tutorial

In this tutorial we will give examples to the usage of all defined resources so far, new sections will be added to every new resource.

Note: To follow along in all sections here make sure to import the library and authenticate using this code:

```
import dopyapi as do
do.authenticate()
```

5.1 List all available regions, images and sizes

When you create Digital Ocean resource, every resource need to be associated with a region, where Digital Ocean data centers exist, there are 9 regions in Digital Ocean they can be fetched using this code:

```
regions = do.Region.list()
for region in regions:
  print(region)
```

The previous code prints all available regions, when you print a Digital Ocean resource object it prints the name of class along with one or two attributes that help you to identify the resource.

An image is used when creating new droplets, it specifies the Operating System found in the droplet when created, the *images* module contains constants for the most popular image names.

To list images use this code:

```
images = do.Image.list()
for image in images:
  print(image)
```

There are other methods for listing images these are *listDistribution()*, *listApplication()*,:meth:*~dopyapi.images.Image.listUser* and *listByTag()*.

When creating droplets we can use the image's object or the image's slug, this will be described later.

Now for tor the sizes, the size of a droplet defines how much resources will be allocated for it such as CPU, RAM and Disk, each droplet will have size associated with it, you can use size objects or sizes slugs which are provided as constants in *sizes* module.

The following code lists some available sizes:

```
sizes = do.Size.list()
for size in sizes:
  print(size)
```

5.2 List SSH Keys and tags

To list all available SSH keys in your account use this code:

```
ssh_keys = do.SSHKey.list()
for ssh_key in ssh_keys:
  print(ssh_key)
```

Tags are associated with resources to help us group similar resources together, if we specify a tag for a new resource and this tag does not exist it is automatically created for us by Digital Ocean.

The following code shows how to list tags:

```
tags = do.Tag.list()
for tag in tags:
  print(tag)
```

To create a new tag use this code:

```
tag = do.Tag()
tag.create(name="web-backend")
```

You can also check all the resources tagged with `web-backend` for example using this code:

```
tag = do.Tag()
tag.name = "web-backend"
print(tag.resources)
```

Here we get a dictionary that contains an attribute for each resource that can be tagged along with counts, an example value is shown bellow:

```
"resources": {
  "count": 5,
  "last_tagged_uri": "https://api.digitalocean.com/v2/images/7555620",
  "droplets": {
    "count": 1,
    "last_tagged_uri": "https://api.digitalocean.com/v2/droplets/3164444"
  },
  "images": {
    "count": 1,
    "last_tagged_uri": "https://api.digitalocean.com/v2/images/7555620"
  },
  "volumes": {
    "count": 1,
```

(continues on next page)

(continued from previous page)

```
    "last_tagged_uri": "https://api.digitalocean.com/v2/volumes/3d80cb72-342b-4aaa-
↪b92e-4e4abb24a933"
  },
  "volume_snapshots": {
    "count": 1,
    "last_tagged_uri": "https://api.digitalocean.com/v2/snapshots/1f6f46e8-6b60-11e9-
↪be4e-0a58ac144519"
  },
  "databases": {
    "count": 1,
    "last_tagged_uri": "https://api.digitalocean.com/v2/databases/b92438f6-ba03-416c-
↪b642-e9236db91976"
  }
}
```

5.3 Create a new droplet

To create a new droplet we need the following:

- name: The name of the droplet.
- image: The image used in the droplet, this could be a slug or *Image* object.
- size: The size used in the droplet, this could be a slug or *Size* object.
- region: The region where the droplet will be created, this could be the region's name or an object of *Region*
- ssh_keys: A list of SSH keys to insert into the droplet, this is optional, you can use either the IDs of keys or *SSHKey* object, this is passed as a list.

Use this code to create the droplet:

```
droplet = do.Droplet()
droplet_data = {
  "name": "d1",
  "image": do.images.ubuntu,
  "region": "ams3",
  "size": do.sizes.small,
  "ssh_keys": do.SSHKey.list()
}
droplet.create(**droplet_data)
```

Droplet actions can be called using methods with the same as action name, and use the same parameters as the action, you can find details in the *Droplet* API.

5.4 Create and list firewalls

To list firewalls use this code:

```
fws = do.Firewall.list()
for fw in fws:
  print(fw)
```

We have shown previously here *Create a new firewall and assign it to a droplet* how to create firewalls.

5.5 Create and list Block Storage volumes

To create a block storage volume we need to specify three required attributes:

- name: The name of the volume.
- size_gigabytes: The size of the volume in Giga Bytes.
- region: The region slug where the volume will be created, you can also use a *Region* object.
- description: This text could be used to describe the volume, it is optional.
- tags: A list of tags to assign to the volume, it could consist of tag names or *Tag* objects, it is optional.

Use this code to create a new tag:

```
volume = do.Volume()
volume_data = {
  "name": "v1",
  "region": "ams3",
  "size_gigabytes": 10,
  "tags": ["db_data"],
  "description": "Store database files"
}
volume.create(**volume_data)
```

To list volumes use this code:

```
volumes = do.Volume.list()
for volume in volumes:
  print(volume)
```

To take a volume snapshot, use this code:

```
snapshot = volume.snapshot(name="s1")
print(snapshot)
```

To attach a volume to a droplet use this code:

```
volume.attach(droplet_id = droplet)
```

5.6 Create and list load balancers

To create a new load balancer we must prepare these attributes:

- name: The name of the load balancer.
- region: The region slug where the load balancer will be created, you can also use the *Region* object.
- forwarding_rules: A list of *ForwardingRule*, each of these objects defines how will the load balancer forward traffic to backend droplets, at least one rule shoul exist.
- sticky_sessions: An object of *StickySession*, which specifies how sessions are handled, this is optional.
- health_check: An object of *HealthCheck*, which specifies how backend droplets are checked for their health, this is optional.
- redirect_http_to_https: A boolean value that determines if HTTP traffic will be redirected to HTTPS by the laod balancer, this defaults to False.

To create a load balancer, use this code:

```
lb = do.LoadBalancer()
forwarding_rule = do.ForwardingRule()
forwarding_rule.entry_protocol = "http"
forwarding_rule.entry_port = 80
forwarding_rule.target_protocol = "http"
forwarding_rule.target_port = 80
sticky_session = do.StickySession()
sticky_session.type = "cookies"
sticky_session.cookie_name = "lb-do"
health_check = do.HealthCheck()
health_check.protocol = "http"
health_check.port = 80
health_check.path = "/check"
lb_data = {
  "name": "lb1",
  "region": "ams3",
  "forwarding_rules": [forwarding_rule],
  "sticky_sessions": sticky_session,
  "health_check": health_check
}
lb.create(**lb_data)
```

Here we used an object of *ForwardingRule* to create a single forwarding_rule and passed a list to the create method, we created an object of *StickySession* to use cookie based sessions, with cookie name set to "lb-do", and lastly we used the *HealthCheck* class to tell the load balancer to use the path */check* for health checks instead of / default.

5.7 Create a floating IP and assign it to a droplet

To create floating IPs we need one of these two:

- A droplet id to assign the IP to it.
- A region slug or *Region* object to reserve the IP for the used region.

Check this code for both methods to create IPs:

```
fp_droplet = do.FloatingIP()
fp_droplet.create(droplet_id = droplet)
fp_region = do.FloatingIP()
fp_region.create(region="ams3")
```

Assigning a floating IP to a droplet is done using the assign method:

```
ac = fp_region.assign(droplet_id = droplet)
print(ac)
```

Here `ac` is an *Action* object.

5.8 Retrieve Balance and Billing information

To get your current available balance use the *Balance* class, and to get your bills use the *BillingHistory* class as follows:

```
balance = do.Balance()
print(balance.json())
bills = do.BillingHistory.list()
for bill in bills:
print(bill.json())
```

5.9 Create and transfer custom images

You can create custom private images in Digital Ocean, use this code to create a custom image with a minimal Ubuntu 18.04 pre-installed:

```
image = do.Image()
image_data = {
  "name": "ubuntu-18-04-minimal",
  "url": "http://cloud-images.ubuntu.com/minimal/releases/bionic/release/ubuntu-18.04-
↪minimal-cloudimg-amd64.img",
  "distribution": do.images.Distribution.ubuntu,
  "region": "ams3",
  "description": "A minimal Ubuntu 18.04 installation",
  "tags": ["custom-image"]
}
image.create(**image_data)
```

After the image is created we can find it using *listUser()* method as shown bellow:

```
images = do.Image.listUser()
for image in images:
  print(image)
```

To transfer this image we can use this code:

```
images = do.Image.listUser()
for image in images:
  if image.name == "ubuntu-18-04-minimal":
      action = image.transfer(region="nyc3")
      print(action)
```

5.10 Create and List VPCs

You can use the *VPC* class to manage VPCs on Digital Ocean, the list class method is used to list all VPCs, to create a new VPC you can use this code:

```
vpc = do.VPC()
vpc_data = {
  "name": "ams3-vpc",
  "region": "ams3",
  "description": "A new VPC in ams3"
}
vpc.create(**vpc_data)
```

In the previous code we did not specify an IP range for the VPC, it was selected automatically by Digital Ocean for us, we can specify our own IP range with adding `ip_range` key to the request, however we must make sure that the range is unique within our account and also must not be smaller than `/24` or larger that `/16`.

5.11 Create domains and domain records

To create a new domain use this code:

```
domain_data = {
  "name": "domain.tld",
  "ip_address": "192.168.12.29"
}
domain = do.Domain()
domain.create(**domain_data)
print(domain)
```

As usual, you can list domains with this code:

```
domains = do.Domain.list()
for domain in domains:
  print(domain)
```

Each domain consists of domain records, these are represented as instances of class *DomainRecord*.

To get the records of a domain use this code:

```
domain_records = do.DomainRecord.list("example.dev")
for record in domain_records:
  print(record.json())
```

Or you can use the domain's object directly to list its records as follows:

```
domain = do.Domain()
domain.name = "example.dev"
records = domain.records()
for record in records:
  print(record.json())
```

You can create a new domain record of type *A* using this code:

```
domain_record = do.DomainRecord("example.dev")
record_data = {
  "type": "A",
  "name": "test",
  "data": "178.12.212.4"
}
domain_record.create(**record_data)
print(domain_record)
```

To delete a domain or domain record just use *delete* method on their objects as usual.

5.12 Create and list database clusters

This library helps you to work with managed databases in Digital Ocean, we will show you here how to create and manage database clusters in Digital Ocean.

To create a new database cluster use this code:

```
db_data = {
  "engine": "mysql",
  "name": "db-mysql1",
  "size": do.sizes.db_tiny,
  "region": "ams3",
  "num_nodes": 1
}
db = do.DatabaseCluster()
db.create(**db_data)
print(db)
```

When creating a new database cluster we need to select the following:

- The engine of cluster: This defines the cluster's type, there are 3 available types: "pg" for PostgreSQL, "mysql" for MySQL and "redis" for Redis.
- The name of the cluster, we need to select a unique name for it.
- The size of cluster: this defines the size of resources reserved for the cluster, you can find available sizes in *sizes* module.
- The region for the cluster, this defines where the cluster's resources will be created.
- The number of nodes: Here we select a number of instances for the database cluster.

You can list database clusters as usual with this code:

```
dbs = do.DatabaseCluster.list()
for db in dbs:
  print(db.json())
```

5.13 Add firewall rules to database clusters

Every database cluster has a set of inbound rules that restricts all soucres from connecting to the cluster except for the specified ones.

The sources can be one of:

- A droplet: Here the type is called *droplet* and the value is the droplet's ID.
- An IP address: Here the type is called *ip_addr* and the value is the IP address in CIDR format.
- A kubernetes cluster: Here the type is *k8s* and the value is the ID of Digital Ocean kubernetes cluster.
- A tag: Here the type is *tag* and the value is the name of tag, all droplets and kubernetes clusters tagged with this tag are automatically allowed in the database cluster.

To create a firewall rule for the database cluster and assign it to the cluster use this code, here we assume that *db* is an instance of *DatabaseCluster*:

```
dbf1 = do.DatabaseFirewall("ip_addr", "178.12.45.4")
dbf2 = do.DatabaseFirewall("tag", "db-allowed")
db.updateFirewall([dbf1, dbf2])
```

In this code we first create two instances of *DatabaseFirewall* to be used in creating the rules, then we use the *updateFirewall()* method to apply the firewalls to the cluster.

To list the firewall rules and make sure they are applied use this code:

```
fws = db.listFirewall()
for fw in fws:
  print(fw)
```

5.14 Configure maintenance window

Each database cluster has a window for maintenance where cluster upgrades might happen, to you can get and set this window with the following code:

```
db.setMaintenanceWindow("friday", "00:00:00")
db.load()
print(db.maintenance_window)
```

The method *setMaintenanceWindow()* is used to configure the maintenance window for the database, here we used the *load()* method to update the values for the object and then print the new maintenance window object.

5.15 Manage Users and Databases

The following code shows how to list, create, reset authentication and delete users:

```
dbs = do.DatabaseCluster.list()
db = dbs[0]
x = db.addUser("mohsen")
print(x)
```

To retrieve an existing user use this code:

```
user = db.getUser("mohsen")
```

To change the user authentication mechanism use this code:

```
x = db.resetAuth("mohsen", "mysql_native_password")
print(x)
```

Listing all database users can be done with this code:

```
users = db.listUsers()
for user in users:
  print(user.json())
```

Deleting a user can be done as follows:

```
x = db.deleteUser("mohsen")
print(x)
```

Managing databases can be done with similar code as shown bellow:

```
dbs = db.listDBS()
for d in dbs:
    print(d)
db.addDB("telegram")
database = db.getDB("telegram")
```

(continues on next page)

(continued from previous page)

```
print(database)
db.deleteDB("telegram")
```

First we list all databases with *listDBS()* method, we then add a new database using *addDB()* method, and get the newly created database using *getDB()* method and finally use *deleteDB()* method to delete a database.

5.16 Manage Connection pools for PostgreSQL database cluster

Use this code to create a connection pool for a PostgreSQL database cluster:

```
pool_data = {
  "name": "con-pool1",
  "db": "defaultdb",
  "user": "doadmin",
  "mode": "transaction",
  "size": 10
}
pool = do.DatabaseConnectionPool(**pool_data)
db.addPool(pool=pool)
# db.addPool(**pool_data)
```

Each pool needs these attributes:

- name: A unique name for the pool.
- db: The database for use with the connection pool.
- user: The name of the user for use with the connection pool.
- mode: The PGBouncer pool mode for the connection pool. The allowed values are session, transaction, and statement.
- size: The size of the PGBouncer connection pool. The total available size for all pools depends on cluster node count and size, the lowest cluster size allows for 25 connections 2 are reserved for control purposes which leaves 23 for the user.

We can create a new instance of *DatabaseConnectionPool* class to create the pool or just pass pool attributes to *addPool()* method to create the pool.

Listing pools and retrieving single pools and deleting them can be done as follows:

```
pools = db.listPools()
for pool in pools:
  print(pool.json())
pool = db.getPool("con-pool1")
db.deletePool("con-pool1")
```

5.17 Manage SQL Mode for MySQL cluster

For MySQL clusters we can manage the used SQL mode as follows:

mode = db.getSqlMode() print(mode) db.setSqlMode("ANSI") mode = db.getSqlMode() print(mode) db.setSqlMode() mode = db.getSqlMode() print(mode)

We use the method *getSqlMode()* to retrieve the current SQL mode, and the method *setSqlMode()* can be used to change it, if no parameters are passed then the mode is reset to default value.

5.18 Manage Eviction policy for Redis clusters

You can get and set the eviction policy using this code:

```
policy = db.getEvPolicy()
print(policy)
db.setEvPolicy("allkeys_lru")
policy = db.getEvPolicy()
print(policy)
```

Allowed values for eviction policy are `noeviction`, `allkeys_lru`, `allkeys_random`, `volatile_lru`, `volatile_random` and `volatile_ttl`.

5.19 Create, update and delete kubernetes clusters

You can use *DOKS* class to manage kubernetes clusters in Digital Ocean.

Use this code to create a new cluster:

```
cluster = do.DOKS()
node_pool = do.NodePool("front-end", "s-1vcpu-2gb", 3)
cluster_data = {
    "name": "doks-test",
    "region": "ams3",
    "version": "1.18",
    "node_pools": [node_pool]
}
cluster.create(**cluster_data)
```

First we create a new instance of *DOKS* and also prepare a Node Pool using *NodePool*, we set the name for the node pool, its size and the number of nodes in it.

After that we prepare the attributes required to create the cluster, these are:

- name: A human readable name for the cluster.
- region: The region where the cluster is created.
- version: The version of kubernetes to be used.
- node_pools: A list of *NodePool* objects, to be created with the cluster.

We can list clusters using this code:

```
dokss = do.DOKS.list()
for doks in dokss:
print(doks.json())
```

We can also update the cluster easily, by updating its dynamic attributes and then calling *save()*:

```
doks.name = "new-cluster-name"
doks.save()
```

To delete the cluster use this code:

```
cluster.delete()
```

5.20 Manage Node Pools for Kubernetes Cluster

We can use *DOKS*, *Node* and *NodePool* to list, add, update and delete nodes and node pools for kubernetes clusters.

The following code lists all node pools for the cluster:

```
node_pools = cluster.listNodePools()
for node_pool in node_pools:
  print(node_pool.getJSON())
```

To get a node pool by ID use *getNodePool()* method, by passing the id value to it, it returns *NodePool* instance.

Add a new node pool with this code:

```
cluster.addNodePool("s-1vcpu-2gb", "new-pool", 3)
```

Delete a node pool with this code:

```
cluster.deleteNodePool(node_pool_id)
```

CHAPTER 6

API Reference

6.1 resource

exception dopyapi.resource.**ClientError**
This exception is raised when the client sends a wrong request

exception dopyapi.resource.**ClientForbiddenError**
This exception is raised when the client is forbidden from accessing Digital Ocean.

exception dopyapi.resource.**DOError**
This exception is raised when an error happens in Digital Ocean side.

class dopyapi.resource.**Resource**(*resource*)
The base class for all managed Digital Ocean resources

In this class we have methods to make API requests using HTTP verbs (GET, POST, DELETE, HEAD and PUT) which are documented in Digital Ocean API documentation, we also use python magic methods to manage instance attributes of managed resources.

auth
An authentication object which holds the used access token and base URL for all API calls.

Type *auth.Auth*

resource
This attribute holds the class of the managed resource, this class must extend Resource class and define these class attributes:

_url: The API endpoint used to manage the resource.

_single: The dictionary index used when fetching single instances.

_plural: The dictionary index used when fetching multiple instances.

_fetch_attrs: A list of attributes that can be used to fetch single instances.

_static_attrs: A list of attributes set by DO API and cannot be changed directly.

_dynamic_attrs: A list of attributes that can be changed and saved or used when creating new instances.

_action_attrs: A list of actions that can be used on this resource.

_delete_attr: An attribute used when deleting instances.

_update_attr: An attribute used when updating instances.

_action_attr: An attribute used when calling actions.

_id_attr: The attribute that hold a unique identifier for the resource.

_resource_type: The type of resource as a string

Type type

action(***kwargs*)

Call an action based on the action_attr and passed arguments

Parameters

- **url** (*str*) – The url used to call the action, if None then the resource's action attribute is used. default None
- **tag_name** (*str*) – The name of tag to use as a query string. default None
- **type** (*str*) – The type of action called.

Returns The action just created.

Return type *Action*

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400, 422, 409 and 429.
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

create(***kwargs*)

This method is used to create a new instance based on arguments.

Returns JSON object from the API

Return type dict

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400, 422, 409 and 429.
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

delete(***kwargs*)

Send a DELETE request to Digital Ocean API

Parameters

- **url** (*str*) – The URL used when sending the request to the API
- **data** (*str*) – The data to send with the request this is optional and it defaults to None.

Returns A dictionary with one key "status" and value "deleted" if status code is 204 or 404

Return type dict

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400 or 422.
- *ClientForbiddenError* – This is raised when the status code is 403

get(*url*, ***kwargs*)
Send a GET request to Digital Ocean API

Parameters **url** (*str*) – The URL to fetch from the API

Returns The dictionary response from the API if status code is 200.

Return type dict

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400 or 422
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

getAction(*action_id*)
This method is used to fetch a single action based on its ID.

Parameters **action_id** (*int*) – The ID of action to fetch.

Returns The action object with the used ID.

Return type *Action*

Raises

- *ClientError* – This is raised if this resource does not support actions or the status code is 400 or 422.
- *DOError* – This is raised when the status code is 500
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

getID()
Return the ID value for the instance.

json()
Return a dictionary of all Digital Ocean attributes for the resource

Returns A dictionary of key/value pairs for the object's attributes.

Return type dict

classmethod list(**args*, ***kwargs*)
This method is used to fetch multiple instances from the API.

Parameters

- **url** (*str*) – The URL used for fetching, it defaults to the defined URL for the resource.
- **page** (*int*) – The number of page in results to fetch default is 1
- **per_page** (*int*) – The number of instances in a single page default is 20

Returns A list of dictionaries from Digital Ocean API.

Return type list

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400 or 422
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

listActions(***kwargs*)
This method is used to list all actions for this instance

Returns A list of action objects

Return type list

Raises

- *ClientError* – This is raised if this resource does not support actions or the status code is 400 or 422.
- *DOError* – This is raised when the status code is 500
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

load()
This method is used to force loading the attributes from the API.

Returns The value for ID attribute.

post(*url*, *data*, ***kwargs*)
Send a POST request to Digital Ocean API

Parameters **url** (*str*) – The URL used when sending the request to the API

Returns The dictionary response from the API if status code is 201 or 202.

Return type dict

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400, 422, 409 and 429.
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

put(*url*, *data*, ***kwargs*)
Send a PUT request to Digital Ocean API

Parameters **url** (*str*) – The URL used when sending the request to the API

Returns

The dictionary response from the API if status code is 204, if the status code is 202 it is {"status": "success"}

Return type dict

Raises

- *DOError* – This is raised when the status code is 500

- *ClientError* – This is raised when the status code is 400 or 422.
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

save(*url=None*)
Update the instance with all values for dynamic attributes

This method calls PUT on the URL for updating the instance, it passes all values for dynamic attributes to prevent any loss of any value for any attribute.

Parameters **url** (*str*) – The url used to send the update request, if it is None then the default URL for the resource is used. default None

Returns The dictionary response from the API if status code is 204.

Return type dict

Raises

- *DOError* – This is raised when the status code is 500
- *ClientError* – This is raised when the status code is 400 or 422.
- *ClientForbiddenError* – This is raised when the status code is 403
- *ResourceNotFoundError* – This is raised when the status code is 404

exception dopyapi.resource.**ResourceNotFoundError**
This exception is raised when we try to access a URL that does not exist.

6.2 1-Click Applications

class dopyapi.clickapps.**ClickApp**(*data=None*)
This class represents 1-Click applications in Digital Ocean.

These are pre-built Droplet images and kubernetes apps already setup for you.

slug
The slug identifier for the 1-Click application.

Type str

type
The type of the 1-Click application, it could be either 'droplet' or 'kubernetes' only so far.

Type str

classmethod list(*type=None*)
This method returns a list of Click Apps as defined by its arguments

Parameters **type** (*str*) – The type of Click Apps to list, if None it fetches all click apps. default None

Returns A list of Click Apps

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422

- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listDroplet()
This method returns a list of Click Apps of type "droplet"

Returns A list of Click Apps of type droplet only

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listKubernetes()
This method returns a list of Click Apps of type kubernetes

Returns A list of Click Apps of type kubernetes

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.3 account

class dopyapi.account.**Account**
Get general information about your account

This class is used to retrieve information about the user's account, these information are saved as instance attributes described below

email
The email of the user

Type str

uuid
A universal unique ID for the user

Type str

droplet_limit
The maximum number of droplets this user can create

Type int

floating_ip_limit
The maximum number of floating IPs this user can create

Type int

email_verified
This checks if the user has verified their email address

Type bool

status
The status of user acount

Type str

status_message
A string that describes the status of user account

Type str

6.4 actions

class dopyapi.actions.**Action**(*data=None*)
A class used to manage actions in DigitalOcean.

This is a general class that can used with all types of actions it is used when getting information about actions.

id
A unique identifier for the action

Type int

status
The status of the action it could be "in-progress", "completed" and "errored"

Type str

type
The type of action, for example "transfer" to represent the state of image transfer action.

Type str

started_at
The time when the action started

Type datetime

completed_at
The time when the action was completed.

Type datetime

resource_id
The unique identifier for the resource associated with this action

Type int

resource_type
The type of resource associated with this action

Type str

region
The region where the action occured.

Type *Region*

region_slug
The region slug where the action occured.

Type str

classmethod list(***kwargs*)
Used to get a list of all actions

Parameters

- **per_page** (*int*) – The number of actions returned in a single page result (defaults to 20)
- **page** (*int*) – The page to be fetched from DigitalOcean (defaults to 1)

Returns A list of actions objects

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

6.5 auth

class dopyapi.auth.**Auth**(*token=None*, *base_url='https://api.digitalocean.com/v2'*)
This class holds authentication information

Here we store the authentication information needed by other classes to access and authenticate to Digital Ocean API, these information are stored in attributes as shown below

token
The authentication token for the Digital Ocean API.

Type str

base_url
The base URL used for all API calls. Defaults (https://api.digitalocean.com/v2)

Type str

Raises *AuthenticationNeeded* – This is raised in case no token is provided and it cannot be found in "DO_TOKEN" environment variable.

exception dopyapi.auth.**AuthenticationNeeded**
This exception is raised when no authentication data is provided.

dopyapi.auth.**authenticate**(*token=None*, *base_url='https://api.digitalocean.com/v2'*)
Store authentication information for the Resource class

The Resource class is base for all Digital Ocean resources, this function creates a new authentication object from the data provided in arguments and assigns it to the Resource class as class attribute

Parameters

- **token** (*str*) – The token used to authenticate to Digital Ocean API. defaults (None)
- **base_url** (*str*) – The URL used for all API calls. defaults (https://api.digitalocean.com/v2)

Raises *AuthenticationNeeded* – This is raised in case no token is provided and it cannot be found in "DO_TOKEN" environment variable.

6.6 bills

class dopyapi.bills.**Balance**
This class holds information about customer's balance

month_to_date_balance
Balance as of the generated_at time. This value includes the account_balance and month_to_date_usage.

Type str

account_balance
Current balance of the customer's most recent billing activity. Does not reflect month_to_date_usage.

Type str

month_to_date_usage
Amount used in the current billing period as of the generated_at time.

Type str

generated_at
The time at which balances were most recently generated.

Type datetime.datetime

class dopyapi.bills.**BillingHistory**(*data=None*)
This class can be used to retrieve the billing history for customers

description
Description of the billing history entry.

Type str

ammount
Amount of the billing history entry.

Type str

invoice_id
ID of the invoice associated with the billing history entry, if applicable.

Type str

invoice_uuid
UUID of the invoice associated with the billing history entry, if applicable.

Type str

date
Time the billing history entry occured.

Type datetime.datetime

type
Type of billing history entry.

Type str

classmethod list(***kwargs*)
This method returns a list of billing history as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all history (defaults 1)
- **per_page** (*int*) – The number of items per a single page (defaults 20)

Returns A list of billing history

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.7 CDNs

class dopyapi.cdns.**CDN**(*data=None*)

This class represents CDN endpoints in Digital Ocean

Here we can create, list, update and delete CDN Endpoints which are used to serve static content from Digital Ocean Spaces to users all around the world.

id

A unique ID that can be used to identify and reference a CDN endpoint.

Type str

origin

The fully qualified domain name (FQDN) for the origin server which provides the content for the CDN. This is currently restricted to a Space.

Type str

endpoint

The fully qualified domain name (FQDN) from which the CDN-backed content is served.

Type str

created_at

The time when the CDN Endpoint was created.

Type str

ttl

The amount of time the content is cached by the CDN's edge servers in seconds.

Type int

certificate_id

The ID of a DigitalOcean managed TLS certificate used for SSL when a custom subdomain is provided.

Type str

custom_domain

The fully qualified domain name (FQDN) of the custom subdomain used with the CDN Endpoint.

Type str

classmethod list(***kwargs*)
This method returns a list of CDN Endpoints as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all CDN Endpoints (defaults 1)
- **per_page** (*int*) – The number of CDN Endpoints per a single page (defaults 20)

Returns A list of CDN Endpoints

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

6.8 Certificates

class dopyapi.certificates.**Certificate**(*data=None*)
This class represents a single certificate in Digital Ocean.

Certificates can be either, custom (created, uploaded and renewed by the user) or managed (Uses let' encrypt free certificates and managed by Digital Ocean entrirely).

id
A unique ID that can be used to identify and reference a certificate.

Type str

name
A unique human-readable name referring to a certificate.

Type str

not_after
A time value given in ISO8601 combined date and time format that represents the certificate's expiration date.

Type datetime.datetime

sha1_fingerprint
A unique identifier generated from the SHA-1 fingerprint of the certificate.

Type str

created_at
A time value given in ISO8601 combined date and time format that represents when the certificate was created.

Type datetime.datetime

dns_names
An array of fully qualified domain names (FQDNs) for which the certificate was issued.

Type list

state
A string representing the current state of the certificate. It may be "pending", "verified", or "error".

Type str

type
A string representing the type of the certificate. The value will be "custom" for a user-uploaded certificate or "lets_encrypt" for one automatically generated with Let's Encrypt.

Type str

create(*name*, *type='lets_encrypt'*, *private_key=None*, *leaf_certificate=None*, *certificate_chain=None*, *dns_names=[]*)
Create a new SSL certificate

We need to pass the name of certificate and some other values according to its type, if type is 'lets_encrypt' you need to use `dns_names` or if the type is 'custom' you need to use `private_key`, `leaf_certificate` and `certificate_chain`.

Parameters

- **name** (*str*) – The name of certificate.
- **type** (*str*) – The type of certificate, it could be either 'lets_encrypt' or 'custom', default is 'lets_encrypt'
- **private_key** (*str*) – The private key for certificate, required when creating a custom certificate. default is None
- **leaf_certificate** (*str*) – The contents of a PEM-formatted public SSL certificate. required with custom certificates. default is None
- **certificate_chain** (*str*) – The full PEM-formatted trust chain between the certificate authority's certificate and your domain's SSL certificate. required with custom certificates. default is None.
- **dns_names** (*list*) – A list of fully qualified domain names (FQDNs) for which the certificate will be issued. The domains must be managed using DigitalOcean's DNS. required for lets_encrypt certificates. default is []

Returns JSON object from the API

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429. or when reqired attributes use default values, based on certificate type.
- `ClientForbiddenError` – This is raised when the domain is not managed in Digital Ocean.
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod list(***kwargs*)
This method returns a list of certificates as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all certificates (defaults 1)
- **per_page** (*int*) – The number of certificates per a single page (defaults 20)

Returns A list of certificates

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.9 Container Registry

class dopyapi.registry.**DockerCredentials**(*cred*)

This class is used to store docker credentials for a docker registry in Digital Ocean.

apply(*file_name='/home/docs/.docker/config.json'*)

Use this function to save the docker credentials to your docker configuration file

Parameters **file_name** (*str*) – The name of the file that contains docker credentials default value is `$HOME/.docker/config.json`

class dopyapi.registry.**Registry**(*data=None*)

This class represents the container registry in your account.

name

The name of the container registry to validate.

Type str

delete()

Send a DELETE request to Digital Ocean API

Parameters

- **url** (*str*) – The URL used when sending the request to the API
- **data** (*str*) – The data to send with the request this is optional and it defaults to None.

Returns A dictionary with one key "status" and value "deleted" if status code is 204 or 404

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403

getDockerCredentials(*read_write=False*, *expiry_seconds=None*)

Get the docker credentials for this registry.

Returns An object that contains the docker credentials.

Return type *DockerCredentials*

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403

- `ResourceNotFoundError` – This is raised when the status code is 404

validate(*name*)
Make sure the passed name is a valid name for docker registry and can be used here.

Parameters **name** (*str*) – The name to validate.

Returns True if name is valid and False otherwise

Return type bool

class dopyapi.registry.**Repository**(*registry_name*, *data=None*)
This class represents the container Repository in your registry.

registry_name
The name of the container registry.

Type str

name
The name of the repository.

Type str

latest_tag
The latest tag of the repository.

Type *RepositoryTag*

tag_count
The number of tags in the repository.

Type int

classmethod list(*registry_name*, ***kwargs*)
Return a list of repositories based on arguments

Parameters

- **registry_name** (*str*) – The name of the Container Registry
- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of repositories in the page (defaults 20)

Returns A list of repository objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listTags(***kwargs*)
Return a list of repositoriy tags based on arguments

Parameters

- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of repository tags in the page (defaults 20)

Returns A list of repository tag objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

class dopyapi.registry.**RepositoryTag**(*registry_name*, *repository_name*, *data=None*)

This class represents a single tgged repository image in your registry.

registry_name

The name of the container registry.

Type str

repository

The name of the repository.

Type str

tag

The name of the tag.

Type str

manifest_digest

The digest of the manifest associated with the tag.

Type str

compressed_size_bytes

The compressed size of the tag in bytes.

Type int

size_bytes

The uncompressed size of the tag in bytes (this size is calculated asynchronously so it may not be immediately available).

Type int

updated_at

The time the tag was last updated.

Type datetime

deleteByDigest()

Delete current repository tag using its manifest digest value.

6.10 Databases

class dopyapi.databases.**DatabaseBackup**(*created_at*, *size_gigabytes*)

This class represents a database cluster backup.

size_gigabytes

The size of the database backup in GBs.

Type float

created_at
A time value given in ISO8601 combined date and time format at which the backup was created.

Type datetime.datetime

class dopyapi.databases.**DatabaseCluster**(*data=None*)
This class represents a single database cluster in Digital Ocean.

The database cluster simplifies database management, it offers these kinds of clusters "PostgreSQL", "MySQL" and "Redis".

id
A unique ID that can be used to identify and reference a database cluster.

Type str

name
A unique, human-readable name referring to a database cluster.

Type str

engine
A slug representing the database engine used for the cluster. The possible values are: "pg" for PostgreSQL, "mysql" for MySQL, and "redis" for Redis.

Type str

version
A string representing the version of the database engine to use for the cluster. If excluded, the specified engine's default version is used. The available versions for PostgreSQL are "10" and "11" defaulting to the later. For MySQL, the only available version is "8". For Redis, the only available version is "5".

Type str

connection
An object containing the information required to connect to the database (see below).

Type *DatabaseConnection*

private_connection
An object containing the information required to connect to the database via the private network (see below).

Type *DatabaseConnection*

users
A list containing objects describing the database's users (see below).

Type list

db_names
A list of strings containing the names of databases created in the database cluster.

Type list

num_nodes
The number of nodes in the database cluster.

Type int

size
The slug identifier representing the size of the nodes in the database cluster.

Type str

region
The slug identifier for the region where the database cluster is located.

Type str

status
A string representing the current status of the database cluster. Possible values include creating, online, resizing, and migrating.

Type str

maintenance_window
An object containing information about any pending maintenance for the database cluster and when it will occur (see below). The used keys for the maintenance windo object are:

day (str): The day of the week on which to apply maintenance updates (e.g. "saturday").

hour (str): The hour in UTC at which maintenance updates will be applied in 24 hour format (e.g. "00:22:00").

pending (bool): A boolean value indicating whether any maintenance is scheduled to be performed in the next window.

description (list): A list of strings, each containing information about a pending maintenance update.

Type dict

created_at
A time value given in ISO8601 combined date and time format that represents when the database cluster was created.

Type datetime

tags
A list of tags that have been applied to the database cluster.

Type list

private_network_uuid
A string specifying the UUID of the VPC to which the database cluster is assigned.

Type str

Connection and Private Connection

These two dictionaries hold keys and values for connection information, used keys are:

uri (str): A connection string in the format accepted by the psql command. This is provided as a convenience and should be able to be constructed by the other attributes.

database (str): The name of the default database.

host (str): The FQDN pointing to the database cluster's current primary node.

port (int): The port on which the database cluster is listening.

user (str): The default user for the database.

password (str): The randomly generated password for the default user.

ssl (bool): A boolean value indicating if the connection should be made over SSL.

addDB(*name*)
Create a new Database in the cluster.

Parameters **name** (*str*) – The name of new database

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429 or if the database cluster type is 'redis'
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

addPool(***kwargs*)

Add a new connection pool to the database cluster if its type is PostgreSQL

You can pass individual pool attributes here or use a *DatabaseConnectionPool* object.

addUser(*name*, *auth_plugin='caching_sha2_password'*)

Add a new user to the database cluster.

Parameters

- **name** (*str*) – The name to give the database user.
- **auth_plugin** (*str*) – A string specifying the authentication method to be used for connections to the MySQL user account. The valid values are "mysql_native_password" or "caching_sha2_password". If excluded when creating a new user, the default for the version of MySQL in use will be used. As of MySQL 8.0, the default is "caching_sha2_password." default caching_sha2_password

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429, or the database cluster is of type 'redis' or if authentication plugin is neither 'caching_sha2_password' nor 'mysql_native_password'.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

createReplica()

Create a new read only replica.

Parameters

- **name** (*str*) – The name to give the read-only replica.
- **region** (Region) – A slug identifier for the region where the read-only replica will be located. If excluded, the replica will be placed in the same region as the cluster.
- **size** (*str*) – A slug identifier representing the size of the node for the read-only replica. The size of the replica must be at least as large as the node size for the database cluster from which it is replicating.
- **tags** (*list*) – A flat list of tag names as strings to apply to the read-only replica after it is created. Tag names can either be existing or new tags.

- **private_network_uuid** (*str*) – A string specifying the UUID of the VPC to which the read-only replica will be assigned. If excluded, the replica will be assigned to your account's default VPC for the region.

deleteDB(*name*)
Delete a database from the cluster.

Parameters **name** (*str*) – The name of database to delete.

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or if the database cluster type is 'redis'.
- ClientForbiddenError – This is raised when the status code is 403

deletePool(*name*)
Delete a Connection Pool from the cluster.

Parameters **name** (*str*) – The name of the connection pool to delete.

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or if the database cluster type is not 'PostgreSQL'.
- ClientForbiddenError – This is raised when the status code is 403

deleteReplica(*name*)
Delete a read only replica by its name.

This method can only be called for mysql and pg clusters. :param name: The name of replica to delete. :type name: str

Returns A dictionary with one key "status" and value "deleted" if status code is 204 or 404

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or when the database cluster engine is redis
- ClientForbiddenError – This is raised when the status code is 403

deleteUser(*name*)
Delete a database user by name.

Parameters **name** (*str*) – The name of database user to delete.

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or when the cluster type is 'redis'
- `ClientForbiddenError` – This is raised when the status code is 403

getDB(*name*)

Retrieve the database from the cluster.

Parameters **name** (*str*) – The name of database to retrieve.

Returns The dictionary response from the API if status code is 200.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster type is 'redis'.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

getEvPolicy()

Retrieve the configured eviction policy for an existing Redis cluster.

Returns The configured eviction policy.

Return type str

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster type is not 'Redis'.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

getPool(*name*)

Retrieve a connection pool from the cluster.

Parameters **name** (*str*) – The name of the connection pool to retrieve.

Returns The connection pool object.

Return type *DatabaseConnectionPool*

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster type is not 'PostgreSQL'.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

getReplica(*name*)
Return a read only replica by its name.

This method can only be called with mysql and pg clusters

Parameters **name** (*str*) – The name of read only replica

Returns The dictionary response from the API, this dictionary has attributes for ready only replicas.

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or when the database cluster engine is redis
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

getSqlMode()
Retrieve the configured SQL mode for mysql cluster.

Returns A string specifying the configured SQL modes for the MySQL cluster.

Return type str

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or if the database cluster type is not 'mysql'.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

getUser(*name*)
Retrieve information for the database user by name.

Parameters **name** (*str*) – The name of database user to retrieve.

Returns An object representing the retrieved user.

Return type *DatabaseUser*

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or database cluster type is 'redis'
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

classmethod list(***kwargs*)
This method returns a list of databases as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all databases (defaults 1)
- **per_page** (*int*) – The number of databases per a single page (defaults 20)

Returns A list of databases

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listBackups()
List database backups for the cluster.

If the cluster is of type 'redis', this will throw *ClientError* exception, otherwise it will return a list of *DatabaseBackup* objects.

Returns A list of *DatabaseBackup* objects.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422, or the database cluster type is 'redis'
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listDBS()
List all databases in the cluster.

Returns A list of dictionaries as returned from the API.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster type is 'redis'.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listFirewall()
Return a list of all Firewall rules for the cluster.

Returns A list of :class::*~dopyapi.databases.DatabaseFirewall* objects.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or when the database cluster engine is redis
- `ClientForbiddenError` – This is raised when the status code is 403

- `ResourceNotFoundError` – This is raised when the status code is 404

listPools()
List all connection pools in the cluster.

Returns A list of *DatabaseConnectionPool* objects.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster type is not 'PostgreSQL'.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listReplicas()
Return a list of all read only replicas.

This method can only be called with mysql and pg clusters

Returns A list of read only replica dictionaries, each dictionary has attributes for ready only replicas.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or when the database cluster engine is redis
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listUsers()
List all database cluster users.

Returns A list of *DatabaseUser* objects.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or database cluster type is 'redis'
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

migrate(*region*)
Migrate the database cluster to a new region.

Parameters **region** (*str*) – The region to migrate to.

Returns

The dictionary response from the API if status code is 204, if the status code is 202 it is {"status": "success"}

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

replicate(*name*, *size*, *region=None*, *tags=[]*, *private_network_uuid=None*)

Replicate the current database cluster to another one, with a different name and size.

If we do not specify a new region then the same region will be used for the database cluster.

Parameters

- **name** (*str*) – The name of the new cluster.
- **size** (`Size`) – The size of new cluster.
- **region** (`Region`) – The region for new cluster, default None.
- **tags** (*list*) – A list of tags for new cluster, default []
- **private_network_uuid** (*str*) – The UUID of private network for new cluster default is None

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429, or if the cluster type is redis.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

resetAuth(*name*, *auth_plugin*)

Change authentication plugin for the database user.

This is only available for mysql clusters.

Parameters

- **name** (*str*) – The name of database user.
- **auth_plugin** (*str*) – The authentication plugin it could be either 'mysql_native_password' or 'caching_sha2_password'

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429, or the database cluster type is not 'mysql' or the authentication plugin is neither 'caching_sha2_password' nor 'mysql_native_password'.

- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

resize(*size*, *num_nodes*)
Change the size of the database cluster.

Parameters

- **size** (*str*) – The new size of the cluster
- **num_nodes** (*int*) – The new number of nodes

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

setEvPolicy(*policy*)
Set the eviction policy for redis clusters

Parameters **policy** (*str*) – A string specifying the desired eviction policy for the Redis cluster. Valid vaules are: *noeviction*, *allkeys_lru*, *allkeys_random*, *volatile_lru*, *volatile_random*, or *volatile_ttl*.

Returns It returns this dictionary {"status": "success"}

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster's type is not redis.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

setMaintenanceWindow(*day*, *hour*)
Set the maintenance window for the database cluster.

Here you need to setup two values, one for the day of the week and the other is for the hour.

Parameters

- **day** (*str*) – The day of week for maintenance, for example: 'friday'
- **hour** (*str*) – The hour of maintenance, for example 23:55

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.

- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

setSqlMode(*mode='ANSI, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION, NO_ZERO_DATE, NO_ZERO_IN_DATE, STRICT_ALL_TABLES'*)

Set SQL Mode for mysql clusters

Parameters **mode** (*str*) – A single string specifying the desired SQL modes for the MySQL cluster separated by commas. default is "ANSI,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION,NO_ZERO_DATE,NO_ZERO_IN

Returns It returns this dictionary {"status": "success"}

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422 or if the database cluster's type is not mysql.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

updateFirewall(*rules*)

Add a new firewall rule to the database cluster.

Here we can pass a list of rules or a single rule that will be added to the cluster.

Parameters **rules** (*list | DatabaseFirewall*) – A list of rules to add.

Returns The dictionary response from the API.

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

waitReady()

Wait untill the cluster is online, this method returns when it is online

class dopyapi.databases.**DatabaseConnection**(*connection*)

This class holds connection information for the database cluster.

These information can be accessed as attributes.

uri

A connection string in the format accepted by the psql command. This is provided as a convenience and should be able to be constructed by the other attributes.

Type str

database

The name of the default database.

Type str

host
The FQDN pointing to the database cluster's current primary node.

Type str

port
The port on which the database cluster is listening.

Type int

user
The default user for the database.

Type str

password
The randomly generated password for the default user.

Type str

ssl
A boolean value indicating if the connection should be made over SSL.

Type bool

class dopyapi.databases.**DatabaseConnectionPool**(*name*, *mode*, *size*, *db*, *user*, *connection=None*, *private_connection=None*)
This class represents connection pool for a PostgreSQL database cluster

Connection pools can be used to allow a database to share its idle connections.

name
A unique name for the connection pool. Must be between 3 and 60 characters.

Type str

mode
The PGBouncer transaction mode for the connection pool. The allowed values are session, transaction, and statement.

Type str

size
The desired size of the PGBouncer connection pool. The maximum allowed size is determined by the size of the cluster's primary node. 25 backend server connections are allowed for every 1GB of RAM. Three are reserved for maintenance. For example, a primary node with 1 GB of RAM allows for a maximum of 22 backend server connections while one with 4 GB would allow for 97. Note that these are shared across all connection pools in a cluster.

Type int

db
The database for use with the connection pool.

Type str

user
The name of the user for use with the connection pool.

Type str

connection
An object containing the information required to access the database using the connection pool.

Type *DatabaseConnection*

private_connection
An object containing the information required to connect to the database using the connection pool via the private network.

Type *DatabaseConnection*

class dopyapi.databases.**DatabaseFirewall**(*type*, *value*)
This class represents a database firewall or inbound source.

It is used to allow access to the firewall from specific sources such as IP addresses, droplets, kubernetes clusters or resources tagged with some tag.

type
The type of resource that the firewall rule allows to access the database cluster. The possible values are: 'droplet', 'k8s', 'ip_addr', or 'tag'

Type str

value
The ID of the specific resource, the name of a tag applied to a group of resources, or the IP address that the firewall rule allows to access the database cluster.

Type str

class dopyapi.databases.**DatabaseUser**(*data*)
A class that represents a user in the database cluster.

user
The user object as returned from the API.

Type dict

name
The name of database user.

Type str

password
The password of the database user.

Type str

role
A string representing the database user's role. The value will be either "primary" or "normal".

Type str

mysql_settings
An object containing addition configuration details for MySQL clusters

Type dict

mysql_settings dictionary has this key

auth_plugin (str): A string specifying the authentication method in use for connections to the MySQL user account. The valid values are "mysql_native_password" or "caching_sha2_password".

6.11 Domains and Domain Records

class dopyapi.domains.**Domain**(*data=None*)
This class represents a single domain in Digital Ocean.

Domain records are only managed by Digital Ocean, the domain still needs to be bought using a domain registrar, and its NS records updated to the ones provided by Digital Ocean.

name
The name of the domain itself. This should follow the standard domain format of domain.TLD. For instance, example.com is a valid domain name.

Type str

ttl
This value is the time to live for the records on this domain, in seconds. This defines the time frame that clients can cache queried information before a refresh should be requested.

Type int

zone_file
This attribute contains the complete contents of the zone file for the selected domain. Individual domain record resources should be used to get more granular control over records. However, this attribute can also be used to get information about the SOA record, which is created automatically and is not accessible as an individual record resource.

Type str

create(*name*, *ip_address=None*)
Create a new domain

We only need to provide the domain's name and optionally and IP address to be assigned to the apex record.

Parameters

- **name** (*str*) – The domain name to add to the DigitalOcean DNS management interface. The name must be unique in DigitalOcean's DNS system. The request will fail if the name has already been taken..
- **ip_address** (*str*) – This optional attribute may contain an IP address. When provided, an A record will be automatically created pointing to the apex domain. default is None

Returns JSON object from the API

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod list(***kwargs*)
This method returns a list of domains as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all domains (defaults 1)
- **per_page** (*int*) – The number of domains per a single page (defaults 20)

Returns A list of domains

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

records(*page=1*, *per_page=20*)
Return a list of *DomainRecord* for this domain.

Parameters

- **page** (*int*) – The page to fetch from all domain records (defaults 1)
- **per_page** (*int*) – The number of domain records per a single page (defaults 20)

Returns A list of domain records

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

class dopyapi.domains.**DomainRecord**(*domain*, *data=None*)
This class represents a single domain record in Digital Ocean.

Each domain record belongs to a single domain.

id
A unique identifier for each domain record.

Type int

type
The type of the DNS record. For example: A, CNAME, TXT, ... You can find a full list bellow.

Type str

name
The host name, alias, or service being defined by the record.

Type str

data
Variable data depending on record type. For example, the "data" value for an A record would be the IPv4 address to which the domain will be mapped. For a CAA record, it would contain the domain name of the CA being granted permission to issue certificates.

Type str

priority
The priority for SRV and MX records.

Type int

port
The port for SRV records.

Type int

ttl
This value is the time to live for the record, in seconds. This defines the time frame that clients can cache queried information before a refresh should be requested.

Type int

weight
The weight for SRV records.

Type int

flags
An unsigned integer between 0-255 used for CAA records.

Type int

tag
The parameter tag for CAA records. Valid values are "issue", "issuewild", or "iodef"

Type str

Record Types:

A: This record type is used to map an IPv4 address to a hostname.

AAAA: This record type is used to map an IPv6 address to a hostname.

CAA: As specified in RFC-6844, this record type can be used to restrict which certificate authorities are permitted to issue certificates for a domain.

CNAME: This record type defines an alias for your canonical hostname (the one defined by an A or AAAA record).

MX: This record type is used to define the mail exchanges used for the domain.

NS: This record type defines the name servers that are used for this zone.

TXT: This record type is used to associate a string of text with a hostname, primarily used for verification.

SRV: This record type specifies the location (hostname and port number) of servers for specific services.

SOA: This record type defines administrative information about the zone. Can only have ttl changed, cannot be deleted

create(*type*, ***kwargs*)
Create a new domain record

Here we pass the record's type first, then we pass a number of arguments according to the records type.

Parameters

- **type** (*str*) – The type of the record A, AAAA, MX, etc....
- **name** (*str*) – The host name, alias, or service being defined by the record. required for A, AAAA, CAA, CNAME, TXT and SRV types.
- **data** (*str*) – Variable data depending on record type. For example, the "data" value for an A record would be the IPv4 address to which the domain will be mapped. For a CAA record, it would contain the domain name of the CA being granted permission to issue certificates. required for A, AAAA, CAA, CNAME, MX, TXT, SRV, NS
- **priority** (*int*) – The priority of the host (for SRV and MX records. null otherwise). required for MX and SRV records.

- **port** (*int*) – The port that the service is accessible on (for SRV records only. null otherwise). reqired for SRV records.
- **ttl** (*int*) – This value is the time to live for the record, in seconds. This defines the time frame that clients can cache queried information before a refresh should be requested. There is a minimum ttl value of 30, unless it is not set. If not set, the default value is the value of the SOA record. For SOA records, defines the time to live for purposes of negative caching. required for SOA records
- **weight** (*int*) – The weight of records with the same priority (for SRV records only. null otherwise). required for SRV records.
- **flags** (*int*) – An unsigned integer between 0-255 used for CAA records. required for CAA records.
- **tag** (*str*) – The parameter tag for CAA records. Valid values are "issue", "issuewild", or "iodef" required for CAA records.

Returns JSON object from the API

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – When type is not supported or no enough data to create the record.
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod list(*domain_name*, ***kwargs*)

This method returns a list of domain records as defined by its arguments

Parameters

- **domain_name** – The name of the domain to fetch records for it
- **page** (*int*) – The page to fetch from all domain records (defaults 1)
- **per_page** (*int*) – The number of domain records per a single page (defaults 20)

Returns A list of domain records

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.12 common

class dopyapi.common.**DOJSONEncoder**(**args*, ***kwargs*)

This class is used to encode Digital Ocean resources as JSON objects

It can be used with *json* module to encode resources.

It is used as follows:

```
with open("droplets.json", "w") as outfile:
    json.dump(data, outfile, cls=do.DOJSONEncoder, sort_keys=True, indent=4)
```

Here data is a list that contains objects of Digital Ocean resources.

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

6.13 droplets

class dopyapi.droplets.**Droplet**(*data=None*)

This class represents a single Droplet in Digital Ocean

You can use this class to create, update, delete and manage droplets on your Digital Ocean account, all droplet actions are available as instance methods and droplet attributes are available too.

id

A unique identifier for the droplet, it is generated when the droplet is created.

Type int

name

A human readable name for the droplet

Type str

memory

memory of the droplet in megabytes

Type int

vcpus

The number of virtual CPUs.

Type int

disk

The size of droplet disk in megabytes.

Type int

locked

A boolean value that tells if the droplet is locked preventing actions by users.

Type bool

created_at
A time object that tells when the droplet was created.

Type datetime.datetime

status
(str): A status string indicating the state of the droplet, it could be ("new", "active", "off", "archive").

backup_ids
An array of backup IDs that have been created for the droplet.

Type list

snapshot_ids
An array of snapshot IDs that have been created for the droplet.

Type list

features
An array of features enabled for the droplet.

Type list

region
A value for the region where the droplet was created.

Type *Region*

image
A value for the base image used to create the droplet

Type *Image*

size
A value for the size object used to create the droplet, this defines the amount of RAM, VCPUS and disk available for the droplet.

Type *Size*

size_slug
A unique slug identifier for the size of this droplet.

Type str

networks
An object that defines all networks connected to the droplet it includes a key of "IPv4" and "IPv6" if enabled, each key has an array of objects that contain network related information such as IP address, netmask and gateway plus more information specific for the network type.

Type dict

kernel
The current kernel for the droplet.

Type dict

next_backup_window
If backups are enabled for the droplet here we will find an object with keys to the start and end times for the backups.

Type dict

tags
An array of tags used when the droplet was created.

Type list

volume_ids
An array of block storage volumes attached to the droplet.

Type list

vpc_uuid
A string specifying the UUID of the VPC to which the Droplet is assigned.

Type str

Supported actions: You can call these actions as methods on *Droplet* objects and return *Action* objects

enable_backups: Used to enable backups for the droplet

disable_backups: Used to disable backups for the droplet

reboot: used to reboot the droplet

power_cycle: Power cycle the droplet

shutdown: Attempt a gracefull shutdown of the droplet

power_off: hard shutdown of the droplet

power_on: power the droplet back on

restore: Restore this droplet to a previous backup, this takes an `image` arg and it should be the ID of a backup for current droplet.

password_reset: Request a password reset for the droplet.

resize: Resize the droplet for a new size, this takes `size` arg it should be the slug identifier for a size, and also a `disk` arg that can be True or False based on whether you want to resize disk as well or not.

rebuild: Rebuild this droplet with a new image, it takes `image` arg for the image that the droplet will use as new base image.

rename: Chnage the name of the droplet, it takes `name` arg.

change_kernel: Change the kernel of this droplet, it takes `kernel` arg which is the unique number of the new kernel to use.

enable_ipv6: Enable IPv6 for the droplet.

enable_private_networking: Enable private networking for the droplet.

snapshot: Take a snapshot for the droplet, it takes `name` arg.

classmethod actionByTagName(*tag_name*, *action*, ***kwargs*)
Execute the action on all droplets with a specific tag.

Parameters

- **tag_name** (*str*) – The name of tag to use.
- **action** (*str*) – The name of the action.

Returns This object represents the action used.

Return type *Action*

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429.
- `ClientForbiddenError` – This is raised when the status code is 403

- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod deleteByTagName(*tag_name*)
Delete all droplets whose tag_name equals *tag_name*

Parameters **tag_name** (*str*) – The name of the tag to delete droplets that match it

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403

getPrivateIP()
Retrieve the private IP address of the droplet if available

This method makes sure that the droplet is active and returns its IP address as a string, if not available return None

Returns The private IP address or None if not available

Return type str

getPublicIP()
Retrieve the public IP address of the droplet

This method makes sure that the droplet is active and returns its IP address as a string

Returns The public IP address

Return type str

getPublicIPv6()
Retrieve the public v6 IP address of the droplet if available

This method makes sure that the droplet is active and returns its IP address as a string, if not available return None

Returns The public IP v6 address or None if not available

Return type str

classmethod list(***kwargs*)
This method returns a list of droplets as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all droplets (defaults 1)
- **per_page** (*int*) – The number of droplets per a single page (defaults 20)

Returns A list of droplets

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403

- `ResourceNotFoundError` – This is raised when the status code is 404

listBackups(***kwargs*)
Return a list of backups for this droplet

Parameters

- **page** (*int*) – The page of backups to return
- **per_page** (*int*) – The number of backups per a single page (defaults 20)

Returns A list of backups, where each one is a dict

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listByTagName(*tag_name*, ***kwargs*)
This method returns a list of droplets that match the tag name

Parameters

- **tag_name** (*str*) – The tag used when fetching droplets
- **page** (*int*) – The page to fetch from all droplets (defaults 1)
- **per_page** (*int*) – The number of droplets per a single page (defaults 20)

Returns A list of droplets

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listDropletNeighbors()
This method returns a list of droplets that are on the same physical server.

The return value will be a list of lists.

Returns A list of droplets IDs

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listKernels(***kwargs*)
Return a list of kernels that can be used with this droplet

Parameters

- **page** (*int*) – The page of kernels to return
- **per_page** (*int*) – The number of kernels per a single page (defaults 20)

Returns A list of kernels, where each kernel is a dict

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

listNeighbors(***kwargs*)
This method returns a list of droplets that are on the same physical server as this one

Parameters

- **page** (*int*) – The page to fetch from all droplets (defaults 1)
- **per_page** (*int*) – The number of droplets per a single page (defaults 20)

Returns A list of droplets

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

listSnapshots(***kwargs*)
Return a list of snapshots for this droplet

Parameters

- **page** (*int*) – The page of snapshots to return
- **per_page** (*int*) – The number of snapshots per a single page (defaults 20)

Returns A list of snapshots

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

waitReady()
Wait until a droplet is ready and running

6.14 firewalls

class dopyapi.firewalls.**Firewall**(*data=None*)
This class represents firewalls in DigitalOcean

You can use this class to create, update, delete and manage firewalls on your Digital Ocean account, all firewall attributes are available too.

id
A unique identifier for the firewall, it is generated when the firewall is created.

Type str

name
A human readable name for the firewall

Type str

pending_changes
A list of dictionaries each containing the fields "droplet_id", "removing", and "status". It is provided to detail exactly which Droplets are having their security policies updated. When empty, all changes have been successfully applied.

Type list

created_at
A time object that tells when the firewall was created.

Type datetime.datetime

status
A status string indicating the state of the firewall, it could be ("waiting", "succeeded", "failed").

Type str

inbound_rules
A list of *InboundRule* objects which specify inbound rules applied in the firewall.

Type list

outbound_rules
A list of *OutboundRule* which specify outbound rules applied in the firewall.

Type list

droplet_ids
A list containing the IDs of the Droplets assigned to the firewall.

Type list

tags
A list containing the names of the Tags assigned to the firewall.

Type list

addDroplets(*ids*)
Add droplets to this Firewall

When adding droplets to a firewall, its rules are applied to traffic that tries to enter the droplet.

Parameters **ids** (*list*) – A list of droplets to be added, if you use a single value it will be converted to a list for you.

Returns JSON object from the API

Return type dictionary

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

addRules(*rules*)

Add rules to the firewall

Parameters **tags** (*list, InboundRule, OutboundRule*) – A list of rules to add, if you use a single object it is converted to a list for you.

Returns The JSON response from the API

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

addTags(*tags*)

Add tags to the firewall

Parameters **tags** (*list, Tag*) – A list of tags to add, it could be tag names or tag objects, if you use a single name or object it is converted to a list for you.

Returns The JSON response from the API

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

create(*name*, *inbound_rules=[]*, *outbound_rules=[]*)

Create a new Firewall

This method is used to create a new Firewall, if no rules are specified then these defaults are used: * An outbound rule that allows ICMP to all destinations. * An outbound rule that allows TCP to all ports and destinations. * An outbound rule that allows UDP to all ports and destinations.

Parameters

- **name** (*str*) – The name of the firewall
- **inbound_rules** (*list*) – A list of *InboundRule* objects. default []

- **outbound_rules** (*list*) – A list of *OutboundRule* objects. default []

Returns JSON object from the API

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

droplets

Return droplet objects which are added to this firewall

Returns A list of droplet objects

Return type list

classmethod list(***kwargs*)

This method returns a list of firewalls as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all firewalls (defaults 1)
- **per_page** (*int*) – The number of firewalls per a single page (defaults 20)

Returns A list of firewalls

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

removeDroplets(*ids*)

Remove droplets from the Firewall

Parameters **ids** (*list, Droplet*) – A list of IDs or droplet objects to remove, you can pass a single ID or droplet here.

Returns JSON object from the API

Return type dictionary

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422.
- ClientForbiddenError – This is raised when the status code is 403

removeRules(*tags*)

Remove rules from the Firewall

Parameters **ids** (*list, InboundRule, OutboundRule*) – A list of IDs or rule objects to remove, you can pass a single ID or rule object here.

Returns JSON object from the API

Return type dictionary

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403

removeTags(*tags*)

Remove tags from the Firewall

Parameters **ids** (*list, Tag*) – A list of IDs or tag objects to remove, you can pass a single ID or tag here.

Returns JSON object from the API

Return type dictionary

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403

class dopyapi.firewalls.**InboundRule**(*protocol='tcp', ports='all', sources=None*)

This class is used to represent an inbound rule in Digital Ocean Firewall

An inbound rule is applied when packets enter the firewall, it specifies what data is allowed in and any other data that does not match any inbound rule for a firewall is discarded and not allowed to reach the resources associated with the firewall.

sources

An object specifying locations from which inbound traffic will be accepted.

Type *Location*

getJSON()

Return a JSON representation of an inbound rule

This representation is used when making API calls.

Returns A dictionary for the inbound rule

Return type dict

class dopyapi.firewalls.**Location**(*addresses=[], droplet_ids=[], load_balancer_uids=[], tags=[]*)

This class represents sources or destinations for firewall inbound and outbound rules.

A location could be a droplet, load balancer, Individual IP address or ranges of them and droplets by tag.

addresses

A list of strings containing the IPv4 addresses, IPv6 addresses, IPv4 CIDRs, and/or IPv6 CIDRs to which the firewall will allow traffic.

Type list

droplet_ids

A list containing the IDs of the Droplets to which the firewall will allow traffic.

Type list

load_balancer_uids
A list containing the IDs of the load balancers to which the firewall will allow traffic.

Type list

tags
A list containing the names of Tags corresponding to groups of Droplets to which the firewall will allow traffic.

Type list

getJSON()
Return a JSON representation of a location object

This representation is used when making API calls.

Returns A dictionary for the location object

Return type dict

class dopyapi.firewalls.**OutboundRule**(*protocol='tcp'*, *ports='all'*, *destinations=None*)
This class is used to represent an outbound rule in Digital Ocean Firewall

An outbound rule is applied when packets leave the firewall, it specifies what data is allowed out and any other data that does not match any outbound rule for a firewall is discarded and not allowed.

destinations
An object specifying locations to which outbound traffic will be accepted.

Type *Location*

getJSON()
Return a JSON representation of an outbound rule

This representation is used when making API calls.

Returns A dictionary for the outbound rule

Return type dict

class dopyapi.firewalls.**Rule**(*protocol='tcp'*, *ports='all'*)
This class is the base class for inbound and outbound rules in Digital Ocean firewalls.

Here we find the protocol and ports attributes for a rule, the rest of attributes can be found in *InboundRule* and *OutboundRule* classes.

protocol
The type of traffic to be allowed. This may be one of "tcp", "udp", or "icmp".

Type str

ports
The ports on which traffic will be allowed specified as a string containing a single port, a range (e.g. "8000-9000"), or "0" when all ports are open for a protocol. For ICMP rules this parameter will always return "0".

Type str

exception dopyapi.firewalls.**RuleError**(**args*, ***kwargs*)

6.15 floating_ips

class dopyapi.floating_ips.**FloatingIP**(*data=None*)
This class represents a single Floating IP in Digital Ocean

You can use this class to create, update, delete and manage floating IPs on your Digital Ocean account, all floating IP actions are available as instance methods and floating IP attributes are available too.

ip
The public IP address of the floating IP. It also serves as its identifier.

Type str

region
The region that the floating IP is reserved to.

Type *Region*

droplet
The Droplet that the floating IP has been assigned to.

Type *Droplet*

Supported actions: You call these actions as methods on *FloatingIP* and return *Action* objects

assign: Used to assign the floating IP to a droplet, it takes a single argument `droplet_id` which is the ID of droplet or a *Droplet* object.

unassign: Used to remove a floating IP from a droplet.

classmethod list(***kwargs*)
This method returns a list of Floating IPs as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all Floating IPs (defaults 1)
- **per_page** (*int*) – The number of Floating IPs per a single page (defaults 20)

Returns A list of Floating IPs

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.16 images

This module is contains the *Image* class to manage images in Digital Ocean, it also has some constants that contain the slug values for the most popular images in Digital Ocean, so you do not have to memorize all the image slugs.

class dopyapi.images.**Distribution**
This class could be used when creating a new private image to specify the distribution for the image.

arch_linux = 'Arch Linux'
Arch Linux

centos = 'CentOS'
CentOS

coreos = 'CoreOS'
CoreOS

debian = 'Debian'
Debian

fedora = 'Fedora'
Fedora

fedora_atomic = 'Fedora Atomic'
Fedora atomic

freebsd = 'FreeBSD'
FreeBSD

gentoo = 'Gentoo'
Gentoo

opensuse = 'openSUSE'
OpenSUSE

rancheros = 'RancherOS'
RancherOS

ubuntu = 'Ubuntu'
Ubuntu

class dopyapi.images.**Image**(*data=None*)
This class represents a single Image in Digital Ocean

This class is used to manage Images in Digital Ocean, it has methods to list all available images, create and delete images too.

id
A numberic ID for the image used in Digital Ocean to identify the image defaults (None)

Type int

name
A human readable name for the image used in User Interfaces. defaults (None)

Type str

type
The type of the image it could be one of the following ("snapshot", "backup", "custom") defaults (None)

Type str

distribution
Here we store the base distribution used in the image. defaults (None)

Type str

slug
A unique string that identifies the image. defaults (None)

Type str

public
This checks if the image is public or not. defaults (None)

Type bool

regions
An array of regions where this image is available. defaults (None)

Type list

min_disk_size
The minimum size in gigabytes needed to create a droplet of this image. defaults (None)

Type int

size_gigabytes
The size of the image in gigabytes. defaults (None)

Type float

description
A description of the image. defaults (None)

Type str

tags
A list of tags for the image. defaults (None)

Type []

status
This string indicates the status of a custom image, it could have one of these values ("NEW", "available", "pending", "deleted"). defaults (None)

Type str

error_message
An error image for the custom image. defaults (None)

Type str

Supported actions: You call these actions as methods on *Image* and return *Action* objects

transfer: This is used to transfer an image to another region, it takes one argument called `region`, it could be the regionss slug or a *Region* object.

convert: This is used to convert an image to a snapshot.

classmethod list(***kwargs*)
Return a list of images based on arguments

Parameters

- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of images in the page (defaults 20)

Returns A list of image objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422

- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listApplication(***kwargs*)
Return a list of application images

Parameters

- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of images in the page (defaults 20)

Returns A list of image objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listByTag(*tag_name*, ***kwargs*)
Return a list of images that match the tag

Parameters

- **tag_name** (*str*) – The name of the tag for the images
- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of images in the page (defaults 20)

Returns A list of image objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listDistribution(***kwargs*)
Return a list of distribution images

Parameters

- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of images in the page (defaults 20)

Returns A list of image objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422

- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listUser(***kwargs*)

Return a list of user private images

Parameters

- **page** (*int*) – The page to fetch (defaults 1)
- **per_page** (*int*) – The number of images in the page (defaults 20)

Returns A list of image objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

dopyapi.images.**caprover_18_04 = 'caprover-18-04'**

CapRover with Ubuntu 18.04

dopyapi.images.**cassandra = 'cassandra'**

Cassandra

dopyapi.images.**centos = 'centos-8-x64'**

Default centos, 8 with 64 bit

dopyapi.images.**centos_6_32 = 'centos-6-x32'**

CentOS 6, 32 bit

dopyapi.images.**centos_6_64 = 'centos-6-x64'**

CentOS 6, 64 bit

dopyapi.images.**centos_7_64 = 'centos-7-x64'**

CentOS 7, 64 bit

dopyapi.images.**centos_8_32 = 'centos-8-x32'**

CentOS 8, 32 bit

dopyapi.images.**centos_8_64 = 'centos-8-x64'**

CentOS 8, 64 bit

dopyapi.images.**coreos_alpha = 'coreos-alpha'**

CoreOS alpha

dopyapi.images.**coreos_beta = 'coreos-beta'**

CoreOS beta

dopyapi.images.**coreos_stable = 'coreos-stable'**

CoreOS stable

dopyapi.images.**debian = 'debian-10-x64'**

Default debian image, 10 64 bit.

dopyapi.images.**debian_10_64 = 'debian-10-x64'**

Debian 10 64 bit

dopyapi.images.**debian_9_64 = 'debian-9-x64'**
Debian 9 64 bit

dopyapi.images.**docker = 'docker-18-04'**
Docker on Ubuntu 18.04

dopyapi.images.**fedora = 'fedora-30-x64'**
Default fedora 30 with 64 bit

dopyapi.images.**fedora_27_64 = 'fedora-27-x64'**
Fedora 27 64 bit

dopyapi.images.**fedora_28_64 = 'fedora-28-x64'**
Fedora 28 64 bit

dopyapi.images.**fedora_28_64_atomic = 'fedora-28-x64-atomic'**
Fedora 28 64 bit, atomic

dopyapi.images.**fedora_30_64 = 'fedora-30-x64'**
Fedora 30 64 bit

dopyapi.images.**freebsd = 'freebsd-12-x64'**
Default FreeBSD image, 12 with 64 bit

dopyapi.images.**freebsd_10_4_64 = 'freebsd-10-4-x64'**
FreeBSD 10.4 64 bit

dopyapi.images.**freebsd_10_4_64_zfs = 'freebsd-10-4-x64-zfs'**
FreeBSD 10.4 64 bit with ZFS

dopyapi.images.**freebsd_11_64_ufs = 'freebsd-11-x64-ufs'**
FreeBSF 11 64 bit with UFS

dopyapi.images.**freebsd_11_64_zfs = 'freebsd-11-x64-zfs'**
FreeBSF 11 64 bit with ZFS

dopyapi.images.**freebsd_12_64 = 'freebsd-12-x64'**
FreeBSF 12 64 bit

dopyapi.images.**freebsd_12_64_zfs = 'freebsd-12-x64-zfs'**
FreeBSF 12 64 bit with ZFS

dopyapi.images.**gitea_18_04 = 'gitea-18-04'**
GitEA with Ubuntu 18.04

dopyapi.images.**rancheros = 'rancheros'**
RancherOS

dopyapi.images.**skaffolder_18_04 = 'skaffolder-18-04'**
Skaffolder with Ubuntu 18.04

dopyapi.images.**ubuntu = 'ubuntu-18-04-x64'**
This creates an Ubuntu 18.04 image by default.

dopyapi.images.**ubuntu_14_04_32 = 'ubuntu-14-04-x32'**
Ubuntu 14.04 32 bit image

dopyapi.images.**ubuntu_14_04_64 = 'ubuntu-14-04-x64'**
Ubuntu 14.04 64 bit image

dopyapi.images.**ubuntu_16_04_32 = 'ubuntu-16-04-x32'**
Ubuntu 16.04 32 bit image

dopyapi.images.**ubuntu_16_04_64 = 'ubuntu-16-04-x64'**
Ubuntu 16.04 64 bit image

dopyapi.images.**ubuntu_18_04_64 = 'ubuntu-18-04-x64'**
Ubuntu 18.04 64 bit image

dopyapi.images.**ubuntu_19_10_64 = 'ubuntu-19-10-x64'**
Ubuntu 19.10 64 bit image

6.17 invoices

class dopyapi.invoices.**Invoice**(*data=None*)
This class represents a single Invoice in Digital Ocean.

An invoice is generated on the first day of each month. An invoice preview is generated daily.

invoice_uuid
The UUID of the invoice. The canonical reference for the invoice.

Type str

amount
Total amount of the invoice, in USD. This will reflect month-to-date usage in the invoice preview.

Type str

invoice_period
Billing period of usage for which the invoice is issued, in YYYY-MM format.

Type str

updated_at
Time the invoice was last updated. This is only included with the invoice preview.

Type datetime

getCSV()
Get a CSV summary of the invoice

Returns CSV data as a string

Return type str

getPDF()
Get a PDF summary of the invoice

Returns A bytes object for the PDF data.

Return type bytes

classmethod list(***kwargs*)
This method returns a list of invoices as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all invoices (defaults 1)
- **per_page** (*int*) – The number of invoices per a single page (defaults 20)

Returns A list of invoices

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

saveCSV(*file=None*)
Save a CSV summary to a file.

Parameters **file** – The name of the file to save CSV data to it, default is {invoice_period}.csv

savePDF(*file=None*)
Save a PDF summary to a file.

Parameters **file** – The name of the file to save CSV data to it, default is {invoice_period}.pdf

class dopyapi.invoices.**InvoiceItem**(*invoice_uuid*, *data=None*)
This class represents a single Invoice item in Digital Ocean.

Invoice Items show details for each invoice, such as product names, their usage time and their price.

product
Name of the product being billed in the invoice item.

Type str

resource_uuid
UUID of the resource billing in the invoice item if available.

Type str

resource_id
ID of the resource billing in the invoice item if available.

Type str

group_description
Description of the invoice item when it is a grouped set of usage, such as DOKS or databases.

Type str

description
Description of the invoice item.

Type str

amount
Billed amount of this invoice item. Billed in USD.

Type str

duration
Duration of time this invoice item was used and subsequently billed.

Type str

duration_unit
Unit of time for duration.

Type str

start_time
Time the invoice item began to be billed for usage.

Type datetime

end_time
Time the invoice item stoped being billed for usage.

Type datetime

project_name
Name of the DigitalOcean Project this resource belongs to.

Type str

classmethod list(*invoice_uuid*, ***kwargs*)
This method returns a list of invoice items as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all invoice items (defaults 1)
- **per_page** (*int*) – The number of invoice items per a single page (defaults 20)

Returns A list of invoice items

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

classmethod listPreview(***kwargs*)
This method returns a list of invoice items for the preview invoice generated daily as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all invoice items (defaults 1)
- **per_page** (*int*) – The number of invoice items per a single page (defaults 20)

Returns A list of invoice items

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

class dopyapi.invoices.**InvoiceSummary**(*invoice_uuid*, *data=None*)
This class represents a single Invoice summary in Digital Ocean.

invoice_uuid
UUID of invoice.

Type str

billing_period
Billing period of usage for which the invoice is issued, in YYYY-MM format.

Type str

amount
Total amount of the invoice, in USD. This will reflect month-to-date usage in the invoice preview.

Type str

user_name
Name of the DigitalOcean customer being invoiced.

Type str

user_billing_address
The billing address of the customer being invoiced.

Type dict

user_company
Company of the DigitalOcean customer being invoiced, if set.

Type str

user_email
Email of the DigitalOcean customer being invoiced.

Type str

product_charges
A summary of the product usage charges contributing to the invoice. This will include an amount, and grouped aggregates by resource type under the items key.

Type dict

overages
A summary of the overages contributing to the invoice.

Type dict

taxes
A summary of the taxes contributing to the invoice.

Type dict

credits_and_adjustments
A summary of the credits and adjustments contributing to the invoice.

Type dict

6.18 Kuberenetes Cluster

A module used to interact with Digital Ocean Kuberenetes Cluster API.

class dopyapi.doks.**DOKS**(*data=None*)
This class represents a single kuberenetes cluster in Digital Ocean.

The kuberenetes cluster simplifies kuberenetes management, it supports these versions of kuberenetes (1.16.14-do.0, 1.17.11-do.0, 1.18.9-do.0).

id
A unique ID that can be used to identify and reference a Kubernetes cluster.

Type str

name
A human-readable name for a Kubernetes cluster.

Type str

endpoint
The base URL of the API server on the Kubernetes master node.

Type str

region
The slug identifier for the region where the Kubernetes cluster is located.

Type str

version
The slug identifier for the version of Kubernetes used for the cluster. If set to a minor version (e.g. "1.14"), the latest version within it will be used (e.g. "1.14.6-do.1"); if set to "latest", the latest published version will be used.

Type str

auto_upgrade
A boolean value indicating whether the cluster will be automatically upgraded to new patch releases during its maintenance window.

Type bool

surge_upgrade
A boolean value indicating whether surge upgrade is enabled/disabled for the cluster. Surge upgrade makes cluster upgrades fast and reliable by bringing up new nodes before destroying the outdated nodes.

Type bool

ipv4
The public IPv4 address of the Kubernetes master node.

Type str

cluster_subnet
The range of IP addresses in the overlay network of the Kubernetes cluster in CIDR notation.

Type str

service_subnet
The range of assignable IP addresses for services running in the Kubernetes cluster in CIDR notation.

Type str

vpc_uuid
A string specifying the UUID of the VPC to which the Kubernetes cluster is assigned.

Type str

tags
An array of tags applied to the Kubernetes cluster. All clusters are automatically tagged "k8s" and "k8s:$K8S_CLUSTER_ID."

Type list

maintenance_policy
An object specifying the maintenance window policy for the Kubernetes cluster (see table below).

Type str

node_pools
An object specifying the details of the worker nodes available to the Kubernetes cluster (see table below).

Type list

created_at
A time value given in ISO8601 combined date and time format that represents when the Kubernetes cluster was created.

Type datetime

updated_at
A time value given in ISO8601 combined date and time format that represents when the Kubernetes cluster was last updated.

Type datetime

status
An object containing a "state" attribute whose value is set to a string indicating the current status of the node. Potential values include running, provisioning, and errored.

Type str

Maintenance Policy

This is a dictionary which defines when cluster maintenance will run, it has the following keys:

start_time (str): The start time in UTC of the maintenance window policy in 24-hour clock format / HH:MM notation (e.g., 15:00).

day (str): The day of the maintenance window policy. May be one of "monday" through "sunday", or "any" to indicate an arbitrary week day.

Node Pools

This is a list of *NodePool*.

addNodePool(*size*, *name*, *count*, *tags=[]*, *labels={}*, *auto_scale=False*, *min_nodes=0*, *max_nodes=0*, *taints=[]*)
Create a new node pool.

Parameters

- **size** (*str*) – The size of Node Pool.
- **name** (*str*) – The name of the Node Pool.
- **count** (*int*) – The number of nodes in the Pool.
- **tags** (*list*) – An array of tags to be assigned to the pool.
- **labels** (*dictionary*) – A dictionary of user defined values assigned to the pool.
- **auto_scale** (*bool*) – A boolean value indicating whether auto-scaling is enabled for this node pool. This requires DOKS versions at least 1.13.10-do.3, 1.14.6-do.3, or 1.15.3-do.3.
- **min_nodes** (*int*) – The minimum number of nodes that this node pool can be auto-scaled to. This will fail validation if the additional nodes will exceed your account droplet limit.
- **max_nodes** (*int*) – The maximum number of nodes that this node pool can be auto-scaled to. This can be 0, but your cluster must contain at least 1 node across all node pools.
- **taints** (*list*) – An array of taints to apply to all nodes in a pool. Taints will automatically be applied to all existing nodes and any subsequent nodes added to the pool. When a taint is removed, it is removed from all nodes in the pool.

Returns A dictionary object for the newly created node pool.

Return type dict

clusterlint(*run_id=None*)
Retrieve clusterlint diagnostic.

If no run_id is provided then the last one is used.

Parameters **run_id** (*str*) – The clusterlint run id to fetch.

clusterlintCheck()
Run a clusterlint on the kuberenetes cluster.

credentials(*expiry_seconds=0*)
Return the credentials of the cluster.

Parameters **expiry_seconds** (*int*) – The expiry of the credentials in seconds, if not set or 0 is used then a default of 7 days is used.

Returns A dictionary object which holds keys and values used to connect to the cluster, it has these keys (server, certificate_authority_data, client_certificate_data , client_key_data, token, expires_at).

Return type dict

deleteNode(*id*, *node_id*, *replace=0*, *skip_drain=0*)
Delete an existing node in a node pool by its ID.

Parameters

- **id** (*str*) – The ID of node pool.
- **node_id** (*str*) – The ID of node to delete.

Returns A dictionary object with one key status.

Return type dict

deleteNodePool(*id*)
Delete an existing node pool by ID.

Parameters **id** (*str*) – The ID of node pool to delete.

Returns A dictionary object of one key status.

Return type dict

getNodePool(*id*)
Get a node pool by ID.

Parameters **id** (*str*) – The ID of node pool to get.

Returns A ndoe pool object.

Return type *NodePool*

Raises *ResourceNotFoundError* – When the id of node pool is not found.

kubeconfig(*expiry_seconds=0*)
Get the kubeconfig of the cluster.

Parameters **expiry_seconds** (*int*) – The expiry of the kubeconfig in seconds, if not set or 0 is used then a default of 7 days is used.

Returns A byte object which contains the kubeconfig used to connect to the cluster.

Return type bytes

classmethod list(***kwargs*)
Return a list of kubernetes clusters as defined by its arguments.

Parameters

- **page** (*int*) – The page to fetch from all clusters (defaults 1)
- **per_page** (*int*) – The number of clusters per a single page (defaults 20)

Returns A list of kubernetes clusters

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

listNodePools()
Return a list of node pools in the cluster.

Returns A list of *NodePool* objects.

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or when the database cluster engine is redis
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

listNodes(*id*)
Return a list of nodes in a pool by ID.

Args: id (str): The ID of node pool to get its nodes.

Returns A list of *Node* objects.

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422 or when the database cluster engine is redis
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

options()
Return an object of available versions and sizes.

updateNodePool(*id*, *name*, *count*, *tags=[]*, *labels={}*, *auto_scale=False*, *min_nodes=0*, *max_nodes=0*, *taints=[]*)
Update an existing node pool by ID.

Parameters

- **id** (*str*) – The ID of node pool to update.
- **name** (*str*) – The name of the Node Pool.
- **count** (*int*) – The number of nodes in the Pool.
- **tags** (*list*) – An array of tags to be assigned to the pool.
- **labels** (*dictionary*) – A dictionary of user defined values assigned to the pool.
- **auto_scale** (*bool*) – A boolean value indicating whether auto-scaling is enabled for this node pool. This requires DOKS versions at least 1.13.10-do.3, 1.14.6-do.3, or 1.15.3-do.3.
- **min_nodes** (*int*) – The minimum number of nodes that this node pool can be auto-scaled to. This will fail validation if the additional nodes will exceed your account droplet limit.
- **max_nodes** (*int*) – The maximum number of nodes that this node pool can be auto-scaled to. This can be 0, but your cluster must contain at least 1 node across all node pools.
- **taints** (*list*) – An array of taints to apply to all nodes in a pool. Taints will automatically be applied to all existing nodes and any subsequent nodes added to the pool. When a taint is removed, it is removed from all nodes in the pool.

Returns A dictionary object with one key status.

Return type dict

Raises *ResourceNotFoundError* – When the node pool is not found.

upgrade(*version*)

Upgrade current kuberenetes cluster to specific version.

verion

The slug identifier for the version of Kubernetes to upgrade to. Use *upgrades()* method for available versions.

Type str

Raises *ResourceNotFoundError* – When the version is not available for upgrade

upgrades()

Return an array of available version upgrades for current cluster.

Returns

A list of dictionaries for available upgardes, each one has these keys:

slug (str): The verion slug used in Digital Ocean.

kubernetes_version (str): The corresponding Digital Ocean service.

Return type list

waitReady()

Wait untill the cluster is online, this method returns when it is online.

dopyapi.doks.**DOKS_V_17 = '1.17.13'**

Kuberenetes Version 17 slug

dopyapi.doks.**DOKS_V_18 = '1.18.10'**

Kuberenetes Version 18 slug

dopyapi.doks.**DOKS_V_19 = '1.19.3'**
Kuberenetes Version 19 slug

class dopyapi.doks.**Node**(*id=None*, *name=None*, *status=None*, *created_at=None*, *updated_at=None*, *droplet_id=None*)
This class represents a single node in the Node Pool.

Each node has these attributes

id
A unique ID that can be used to identify and reference the node.

Type str

name
An automatically generated, human-readable name for the node.

Type str

status
An object containing a "state" attribute whose value is set to a string indicating the current status of the node. Potentialvalues include running, provisioning, and errored.

Type dict

created_at
A time value given in ISO8601 combined date and time format that represents when the node was created.

Type datetime

updated_at
A time value given in ISO8601 combined date and time format that represents when the node was created.

Type datetime

class dopyapi.doks.**NodePool**(*name*, *size*, *count*, *labels={}*, *tags=[]*, *auto_scale=False*, *min_nodes=0*, *max_nodes=0*, *nodes=[]*, *id=None*, *taints=[]*)
This class represents a pool of ndoes for kubernetes clusters.

Each pool of nodes defines a number of nodes with a specific size name, labels and auto_scale attribute.

size
The slug identifier for the type of Droplet to be used as workers in the node pool.

Type str

name
A human-readable name for the node pool.

Type str

count
The number of Droplet instances in the node pool.

Type int

labels
An object containing a set of Kubernetes labels. The keys are user-defined.

Type dict

auto_scale
A boolean value indicating whether auto-scaling is enabled for this node pool. This requires DOKS versions at least 1.13.10-do.3, 1.14.6-do.3, or 1.15.3-do.3.

Type bool

min_nodes
The minimum number of nodes that this node pool can be auto-scaled to. This will fail validation if the additional nodes will exceed your account droplet limit.

Type int

max_nodes
The maximum number of nodes that this node pool can be auto-scaled to. This can be 0, but your cluster must contain at least 1 node across all node pools.

Type int

getJSON()
Return JSON representation of *NodePool*.

6.19 load balancers

class dopyapi.loadbalancers.**ForwardingRule**(*entry_protocol='http'*, *entry_port=80*, *target_protocol='http'*, *target_port=80*, *certificate_id=''*, *tls_passthrough=False*)
This class represents a single forwarding rule for Load Balancers

These rules specify how traffic is routed from load balancer to internal droplets assigned for the load balancer, it tells type of traffic it accepts, port and how to send traffic to the droplet, it also tells whether SSL traffic is terminated at the load balancer or the droplets assigned to it.

entry_protocol
The protocol used for traffic to the load balancer. The possible values are: "http", "https", "http2", or "tcp". (default http)

Type str

entry_port
An integer representing the port on which the load balancer instance will listen. (default 80)

Type int

target_protocol
The protocol used for traffic from the load balancer to the backend Droplets. The possible values are: "http", "https", "http2", or "tcp". (default http)

Type str

target_port
An integer representing the port on the backend Droplets to which the load balancer will send traffic. (default 80)

Type int

certificate_id
The ID of the TLS certificate used for SSL termination if enabled.

Type str

tls_passthrough
A boolean value indicating whether SSL encrypted traffic will be passed through to the backend Droplets. (default true)

Type bool

getJSON()
Return the JSON representation of a forwarding rule.

This will be used when sending API requests to create a load balancer.

Returns A dictionary of key/value pairs for the rule's attributes.

Return type dict

class dopyapi.loadbalancers.**HealthCheck**(*protocol='http'*, *port=80*, *path='/'*, *check_interval_seconds=10*, *response_timeout_seconds=5*, *healthy_threshold=5*, *unhealthy_threshold=3*)
This class represents health check objects for Digital Ocean Load Balancer

The health check is used to tell if a droplet is responding or not. The load balancer automatically stops sending traffic to unhealthy droplets.

protocol
The protocol used for health checks sent to the backend Droplets. The possible values are "http" or "tcp".

Type str

port
An integer representing the port on the backend Droplets on which the health check will attempt a connection.

Type int

path
The path on the backend Droplets to which the load balancer instance will send a request.

Type str

check_interval_seconds
The number of seconds between between two consecutive health checks.

Type int

response_timeout_seconds
The number of seconds the load balancer instance will wait for a response until marking a health check as failed.

Type int

unhealthy_threshold
The number of times a health check must fail for a backend Droplet to be marked "unhealthy" and be removed from the pool.

Type int

healthy_threshold
The number of times a health check must pass for a backend Droplet to be marked "healthy" and be re-added to the pool.

Type int

class dopyapi.loadbalancers.**LoadBalancer**(*data=None*)
This class is used to manage Load Balancer in Digital Ocean.

You can use this class to create, update and delete load balancers and assign droplets to them, also specify their forwarding rules.

id
A unique ID that can be used to identify and reference a load balancer.

Type str

name
A human-readable name for a load balancer instance.

Type str

ip
An attribute containing the public-facing IP address of the load balancer.

Type str

alogrithm
The load balancing algorithm used to determine which backend Droplet will be selected by a client. It must be either "round_robin" or "least_connections".

Type str

status
A status string indicating the current state of the load balancer. This can be "new", "active", or "errored".

Type str

created_at
A time value that represents when the load balancer was created.

Type datetime.datetime

forwarding_rules
A list of objects specifying the forwarding rules for a load balancer.

Type list

health_checks
An object specifying health check settings for the load balancer.

Type *HealthCheck*

sticky_sessions
An object specifying sticky sessions settings for the load balancer.

Type *StickySession*

region
The region where the load balancer instance is located.

Type *Region*

tag
The name of a Droplet tag corresponding to Droplets assigned to the load balancer.

Type str

droplet_ids
A list containing the IDs of the Droplets assigned to the load balancer.

Type list

redirect_http_to_https
A boolean value indicating whether HTTP requests to the load balancer on port 80 will be redirected to HTTPS on port 443.

Type bool

enable_proxy_protocol
A boolean value indicating whether PROXY Protocol is in use.

Type bool

addDroplets(*droplets*)
Add new droplets to the load balancer.

If the load balancer was created with a tag attribute, then this method will throw an error because you cannot add droplets to load balancers with a tag attribute, droplets in this case are added automatically when you tag a droplet with this tag, you can pass a single droplet object, or a list of droplet objects, you can also pass IDs instead of objects.

Parameters **droplets** (*list*) – A list of droplets to add, you can pass a single droplet and it will be converted to a list.

Returns JSON object from the API

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

addRules(*rules*)
Add new rules to the load balancer.

Use this method to update the forwarding rules of a load balancer.

Parameters **rules** (*list*) – A list of rules to add, you can pass a single *ForwardingRule* object and it will be converted to a list.

Returns JSON object from the API

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

create(***kwargs*)
Create a new load balancer

If no forwarding rule is specified then a default one that forwards HTTP traffic on port 80 to port 80 without SSL is used.

Returns JSON object from the API

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

deleteDroplets(*droplets*)
Delete droplets from a load balancer.

Parameters **droplets** (*list*) – A list of droplets to delete, you can pass a single droplet and it will be converted to a list.

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422.
- ClientForbiddenError – This is raised when the status code is 403

deleteRules(*rules*)
Delete rules from a load balancer.

Parameters **rules** (*list*) – A list of rules to delete, you can pass a single *ForwardingRule* object and it will be converted to a list.

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422.
- ClientForbiddenError – This is raised when the status code is 403

classmethod list(***kwargs*)
This method returns a list of load balancers as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all load balancers (defaults 1)
- **per_page** (*int*) – The number of load balancers per a single page (defaults 20)

Returns A list of load balancers

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

class dopyapi.loadbalancers.**StickySession**(*type='none'*, *cookie_name='do-lb'*, *cookie_ttl_seconds=60*)
A class to represent sticky sessions used in Digital Ocean Load Balancer

type
An attribute indicating how and if requests from a client will be persistently served by the same backend Droplet. The possible values are "cookies" or "none". If not specified, the default value is "none".

Type str

cookie_name
The name to be used for the cookie sent to the client. This attribute is required when using "cookies" for the sticky sessions type.

Type str

cookie_ttl_seconds
The number of seconds until the cookie set by the load balancer expires. This attribute is required when using "cookies" for the sticky sessions type.

Type int

6.20 projects

class dopyapi.projects.**Project**(*data=None*)
This class represents a project in Digital Ocean.

A project allows you to organize your resources in groups that fit the applications you run on Digital Ocean.

id
The unique universal identifier of this project.

Type str

owner_uuid
The unique universal identifier of the project owner.

Type str

owner_id
The integer id of the project owner.

Type int

name
The human-readable name for the project.

Type str

description
An optional description text for the project.

Type str

purpose
The purpose of the project, it can have one of these values ("Just trying out DigitalOcean", "Class project / Educational purposes" , "Website or blog", "Web Application", "Service or API", "Mobile Application" , "Machine learning / AI / Data processing", "IoT", "Operational / Developer tooling") if you use a valume other than these it will be stored as "Other: your custom purpose".

Type str

environment
The environment for project resources, it can have one of these values ("Development", "Staging", "Production").

Type str

is_default
If true, all resources will be added to this project if no project is specified.

Type bool

created_at
The time when the project was created.

Type datetime

updated_at
The time when the project was updated.

Type datetime

classmethod getDefault()
Return the default project objects

Returns The Project for the default project

Return type *Project*

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

classmethod list(**kwargs*)
Return a list of project instances.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of project instances.

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

class dopyapi.projects.**Purpose**
A class that contains valid values for the project's purpose attribute.

ai = 'Machine learning / AI / Data processing'
A project for Artificial Intelligence purposes.

api = 'Service or API'
A project to host an API.

blog = 'Website or blog'
A project for a blog or website.

education = 'Class project / Educational purposes'
A project for educational purposes.

iot = 'IoT'
A project for an IOT platform.

mobile = 'Mobile Application'
A project for a mobile application.

tools = 'Operational / Developer tooling'
A project for Developer and Operational tools

trying = 'Just trying out DigitalOcean'
A project to try Digital Ocean services.

web = 'Web Application'
A project to host a web application.

6.21 regions

class dopyapi.regions.**Region**(*data=None*)
This class represents a single region in Digital Ocean

A region represents a dataceneter where droplets can be created and images can be transferred.

slug
A human readable string that can be used as a unique identifier for each region.

Type str

name
The display name for the region

Type str

sizes
A list of size slugs that are available for this region

Type list

available
A boolean that checks if the region is available or not

Type bool

features
An array of features available for this region

Type list

classmethod list(***kwargs*)
Return a list of regions based on arguments

Parameters

- **page** (*int*) – The page to return
- **per_page** (*int*) – The number of regions in a single page

Returns A list of region objects

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403

- `ResourceNotFoundError` – This is raised when the status code is 404

6.22 sizes

This module contains the *`Size`* class to manage all available sizes in Digital Ocean used when creating droplets, it also has a set of constants which contain the values for size slugs, so you do not need to memorize all of these slugs when creating a new droplet.

class `dopyapi.sizes.`**Size**(*data=None*)

This class represents sizes in DO which are used when creating droplets.

A size includes the amount of RAM, Virtual CPUs, disk and transfer available for a droplet once created using it.

slug

A human-readable string that is used to uniquely identify each size.

Type str

available

Whether this size is available for droplet creation or not.

Type bool

transfer

The amount of bandwidth transfer available for droplets of this size.

Type float

price_monthly

The monthly cost for this size in US dollars.

Type float

price_hourly

The hourly cost for this size in US dollars.

Type float

memory

The RAM available for this size.

Type int

vcpus

The Virtual CPUs available for this size.

Type int

disk

The amount of disk space available for this size.

Type int

regions

A list containing the region slugs where this size is available for Droplet creates.

Type list

classmethod list(***kwargs*)

Return a list of size instances.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of size instances in a single page. (default 20)

Returns A list of sizes instances.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

`dopyapi.sizes.`**`big = 's-4vcpu-8gb'`**
A big size, 4vCPU, 8 GB RAM

`dopyapi.sizes.`**`c_2_4 = 'c-2'`**
2 vCPU, 4 GB RAM

`dopyapi.sizes.`**`c_4_8 = 'c-4'`**
4 vCPU, 8 GB RAM

`dopyapi.sizes.`**`db_16_64 = 'db-s-16vcpu-64gb'`**
16 vCPU 64 GB RAM, 1.12 TB HD

Type Database size

`dopyapi.sizes.`**`db_1_1 = 'db-s-1vcpu-1gb'`**
1 vCPU 1 GB RAM, 10 GB HD

Type Database size

`dopyapi.sizes.`**`db_1_2 = 'db-s-1vcpu-2gb'`**
1 vCPU 2 GB RAM, 25 GB HD

Type Database size

`dopyapi.sizes.`**`db_2_4 = 'db-s-2vcpu-4gb'`**
2 vCPU 4 GB RAM, 38 GB HD

Type Database size

`dopyapi.sizes.`**`db_4_8 = 'db-s-4vcpu-8gb'`**
4 vCPU 8 GB RAM, 115 GB HD

Type Database size

`dopyapi.sizes.`**`db_6_16 = 'db-s-6vcpu-16gb'`**
6 vCPU 16 GB RAM, 270 GB HD

Type Database size

`dopyapi.sizes.`**`db_8_32 = 'db-s-8vcpu-32gb'`**
8 vCPU 32 GB RAM, 580 GB HD

Type Database size

`dopyapi.sizes.`**`db_large = 'db-s-4vcpu-8gb'`**
4 vCPU 8 GB RAM, 115 GB HD

Type Large database size

dopyapi.sizes.**db_medium = 'db-s-2vcpu-4gb'**
2 vCPU 4 GB RAM, 38 GB HD

Type Medium database size

dopyapi.sizes.**db_small = 'db-s-1vcpu-2gb'**
1 vCPU 2 GB RAM, 25 GB HD

Type Small database size

dopyapi.sizes.**db_tiny = 'db-s-1vcpu-1gb'**
1 vCPU 1 GB RAM, 10 GB HD

Type Tiny database size

dopyapi.sizes.**db_xlarge = 'db-s-6vcpu-16gb'**
6 vCPU 16 GB RAM, 270 GB HD

Type X large database size

dopyapi.sizes.**db_xxlarge = 'db-s-8vcpu-32gb'**
8 vCPU 32 GB RAM, 580 GB HD

Type XX large database size

dopyapi.sizes.**db_xxxlarge = 'db-s-16vcpu-64gb'**
16 vCPU 64 GB RAM, 1.12 TB HD

Type XXX large database size

dopyapi.sizes.**g_2_8 = 'g-2vcpu-8gb'**
2 vCPU, 8 GB RAM

dopyapi.sizes.**gd_2_8 = 'gd-2vcpu-8gb'**
2 vCPU, 8 GB RAM

dopyapi.sizes.**large = 's-6vcpu-16gb'**
A large size, 6vCPU, 16 GB RAM

dopyapi.sizes.**m_1_8 = 'm-1vcpu-8gb'**
1 vCPU, 8 GB RAM

dopyapi.sizes.**m_2_16 = 'm-16gb'**
2 vCPU, 16 GB RAM

dopyapi.sizes.**medium = 's-2vcpu-4gb'**
A medium size, 2vCPU, 4 GB RAM

dopyapi.sizes.**s_1_1 = 's-1vcpu-1gb'**
1 vCPU, 1 GB RAM

dopyapi.sizes.**s_1_2 = 's-1vcpu-2gb'**
1 vCPU, 2 GB RAM

dopyapi.sizes.**s_1_3 = 's-1vcpu-3gb'**
1 vCPU, 3 GB RAM

dopyapi.sizes.**s_2_2 = 's-2vcpu-2gb'**
2 vCPU, 2 GB RAM

dopyapi.sizes.**s_2_4 = 's-2vcpu-4gb'**
2 vCPU, 4 GB RAM

dopyapi.sizes.**s_3_1 = 's-3vcpu-1gb'**
3 vCPU, 1 GB RAM

dopyapi.sizes.**s_4_8 = 's-4vcpu-8gb'**
4 vCPU, 8 GB RAM

dopyapi.sizes.**s_6_16 = 's-6vcpu-16gb'**
6 vCPU, 16 GB RAM

dopyapi.sizes.**small = 's-1vcpu-3gb'**
A small size, 1vCPU, 3 GB RAM

dopyapi.sizes.**t_0_1 = '512mb'**
1 vCPU, 512 MB RAM

dopyapi.sizes.**t_1_1 = '1gb'**
1 vCPU, 1 GB RAM

dopyapi.sizes.**t_2_2 = '2gb'**
2 vCPU, 2 GB RAM

dopyapi.sizes.**t_2_4 = '4gb'**
2 vCPU, 4 GB RAM

dopyapi.sizes.**t_4_8 = '8gb'**
4 vCPU, 8 GB RAM

dopyapi.sizes.**tiny = 's-1vcpu-1gb'**
A tiny size, 1vCPU, 1 GB RAM

6.23 snapshots

class dopyapi.snapshots.**Snapshot**(*data=None*)
This class represents snapshots in Digital Ocean.

Each snapshot is a saved image from a droplet or a block storage volume, the resource_type attribute defines if the snapshot is for a droplet or volume.

id
The unique identifier for the snapshot.

Type str

name
A human-readable name for the snapshot.

Type str

created_at
The date where the snapshot was created.

Type datetime

regions
A list of region slugs that the image is available in.

Type list

resource_id
A unique identifier for the resource that the snapshot is associated with.

Type str

resource_type
The type of resource for this snapshot.

Type str

min_disk_size
The minimum size in GB required for a volume or droplet to use this snapshot.

Type int

size_gigabytes
The size of snapshot.

Type float

tags
A list of tags for the snapshot.

Type list

classmethod list(***kwargs*)
Return a list of snapshot instances.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of snapshot instances.

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

classmethod listDropletSnapshots(***kwargs*)
Return a list of droplet snapshots.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of droplet snapshots.

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

classmethod listVolumeSnapshots(***kwargs*)
Return a list of volume snapshots.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)

- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of volume snapshots.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.24 SSH Keys

class dopyapi.sshkeys.**SSHKey**(*data=None*)

This class represents SSHKeys in Digital Ocean.

SSHKeys are used to embed public keys at droplet creation.

id

A unique identifier for the key.

Type int

fingerprint

The fingerprint value generated from the public key.

Type str

public_key

The entire public key as a string.

Type str

name

A human readable name for the key.

Type str

classmethod list(***kwargs*)

Return a list of SSHKey instances.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of SSHKey instances.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.25 Tags

class dopyapi.tags.**Tag**(*data=None*)
This class represents tags on Digital Ocean.

A tag is applied on resources and helps to group them and facilates lookups and actions on them.

name
A name for the tag.

Type str

resources
An object that contains keys and values for all resources tagged with this tag with count and last_tagged_uri attribute.

Type dictionary

classmethod list(***kwargs*)
Return a list of tag instances.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of tag instances.

Return type list

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

tag(*resources*)
Tag resources with this tag.

Parameters **resources** – A list of objects that represents Digital Ocean resources.

Returns The response from Digital Ocean API.

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400, 422, 409 and 429.
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

unTag(*resources*)
remove this tag from resources with.

Parameters **resources** – A list of objects that represents Digital Ocean resources.

Returns An object with key of "status" and value "deleted".

Return type dict

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422.
- `ClientForbiddenError` – This is raised when the status code is 403

6.26 Volumes

class dopyapi.volumes.**Volume**(*data=None*)

This class represents a single Block Storage volume in Digital Ocean.

A volume appears as locally attached disk to a droplet that can be formatted by the operating system, it can have sizes from 1GB up to 16TB.

id

The unique identifier for the Block Storage volume.

Type str

region

The region where the volume is located.

Type *Region*

droplet_ids

A list that contains the droplet IDs for droplets where this volume is attached to, so far a volume can be attached only to a single droplet.

Type list

name

A human readable name for the volume.

Type str

description

An optional description for the volume.

Type str

size_gigabytes

The size of volume in GB.

Type int

created_at

The time when the volume was created.

Type datetime

filesystem_type

The type of filesystem currently in-use on the volume.

Type str

filesystem_label

The label currently applied to the filesystem.

Type str

tags

A list of Tags the volume has been tagged with.

Type list

classmethod delete_by_name(*name*, *region*)
Delete a volume by name and region name

This method is used to delete a volume by its name and in which region it exists.

Parameters

- **name** (*str*) – The name of the volume.
- **region** (*str, Region*) – The name of region or the region object.

Returns A dictionary with one key "status" and value "deleted".

Return type dict

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422.
- ClientForbiddenError – This is raised when the status code is 403

classmethod get_by_name(*name*, *region*)
Get a volume by specifying a region and name

This method will return a volume with the given name and the used region, volumes with the same name could exist in different regions.

Parameters

- **name** (*str*) – The name of the volume.
- **region** (*str, Region*) – The name of region or the region object.

Returns The volume object found

Return type *Volume*

Raises

- DOError – This is raised when the status code is 500
- ClientError – This is raised when the status code is 400 or 422
- ClientForbiddenError – This is raised when the status code is 403
- ResourceNotFoundError – This is raised when the status code is 404

classmethod list(***kwargs*)
Return a list of volume instances.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of volume instances.

Return type list

Raises

- DOError – This is raised when the status code is 500

- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

classmethod listByName(*name*, ***kwargs*)
Return a list of volume instances by name.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)
- **name** (*str*) – The pattern for volumes name.

Returns A list of volume instances by name.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listSnapshots(***kwargs*)
Return a list of volume snapshots.

Parameters

- **page** (*int*) – The page we want to fetch. (default 1)
- **per_page** (*int*) – The number of snapshot instances in a single page. (default 20)

Returns A list of volume snapshots.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

snapshot(*name*, *tags=[]*)
Take a snapshot of a volume.

This method is used to take a volume snapshot.

Parameters

- **name** (*str*) – The name of snapshot.
- **tags** (*list*) – A list of tags for the snapshot (default [])

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400, 422, 409 and 429.

- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

6.27 VPC

class dopyapi.vpcs.**VPC**(*data=None*)

This class is used to manage Virtual Private Clouds (VPCs) in Digital Ocean.

VPCs allow users to create separate private networks for their resources where resources in one VPC cannot communicate with resources in another VPC.

id

A unique ID that can be used to identify and reference the VPC.

Type str

urn

The uniform resource name (URN) for the VPC.

Type str

name

The name of the VPC. Must be unique and may only contain alphanumeric characters, dashes, and periods.

Type str

region

The slug identifier for the region where the VPC will be created.

Type str

ip_range

The range of IP addresses in the VPC in CIDR notation.

Type str

description

A free-form text field for describing the VPC's purpose. It may be a maximum of 255 characters.

Type str

default

A boolean value indicating whether or not the VPC is the default one for the region.

Type bool

created_at

A time value given in ISO8601 combined date and time format.

Type datetime.datetime

classmethod list(***kwargs*)

This method returns a list of VPCs as defined by its arguments

Parameters

- **page** (*int*) – The page to fetch from all VPCs (defaults 1)
- **per_page** (*int*) – The number of VPCs per a single page (defaults 20)

Returns A list of VPC objects

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

listMembers()

Return a list of members with this VPC

The list contains dictionaries each one with these keys:

urn: The Uniform Resource Name for the resource used.

name: The name of resource

created_at: A time value given in ISO8601 combined date and time format that represents when the resource was created.

Parameters

- **page** (*int*) – The page to fetch from all VPCs (defaults 1)
- **per_page** (*int*) – The number of VPCs per a single page (defaults 20)

Returns A list of dictionaries.

Return type list

Raises

- `DOError` – This is raised when the status code is 500
- `ClientError` – This is raised when the status code is 400 or 422
- `ClientForbiddenError` – This is raised when the status code is 403
- `ResourceNotFoundError` – This is raised when the status code is 404

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z