pycloudlib Documentation

Joshua Powers

Mar 23, 2023

Clouds

Python library to launch, interact, and snapshot cloud instances

CHAPTER 1

Documentation

Use the links in the table of contents to find:

- Cloud specific guides and documentation
- API documentation
- How to contribute to the project

CHAPTER 2

Install

Install directly from PyPI:

```
pip3 install pycloudlib
```

Project's requirements.txt file can include pycloudlib as a dependency. Check out the pip documentation for instructions on how to include a particular version or git hash.

Install from latest master:

```
git clone https://git.launchpad.net/pycloudlib
cd pycloudlib
python3 setup.py install
```

CHAPTER 3

Usage

The library exports each cloud with a standard set of functions for operating on instances, snapshots, and images. There are also cloud specific operations that allow additional operations.

See the examples directory or the online documentation for more information.

CHAPTER 4

Bugs

File bugs on Launchpad under the pycloudlib project.

CHAPTER 5

Contact

If you come up with any questions or are looking to contact developers please use the pycloudlib-devs@lists.launchpad.net list.

5.1 Azure

The following page documents the Azure cloud integration in pycloudlib.

5.1.1 Credentials

To access Azure requires users to have four different keys:

- client id
- client secret id
- tenant id
- subscription id

These should be set in pycloudlib.toml.

Azure login (Deprecated)

By using the Azure CLI, you can login into your Azure account through it. Once you logged in, the CLI will create folder in your home directory which will contain all of the necessary information to use the API. To login into you Azure using the CLI, just run the following command:

```
az login
```

Passed Directly (Deprecated)

All of these four credentials can also be provided directly when initializing the Azure object:

```
azure = pycloudlib.Azure(
    client_id='ID_VALUE',
    client_secret_id='ID_VALUE',
    tenant_id='ID_VALUE',
    subscription_id='ID_VALUE',
)
```

This way we can create different Azure instances with different configurations.

5.1.2 SSH Keys

Azure requires an SSH key to be uploaded before using it. See the SSH Key page for more details.

5.1.3 Image Lookup

To find latest daily Azure image for a release of Ubuntu:

```
azure.daily_image('xenial')
"Canonical:UbuntuServer:16.04-DAILY-LTS"
```

The return Azure image can then be used for launching instances.

5.1.4 Instances

Launching an instance requires at a minimum an Azure image.

```
inst_0 = azure.launch('Canonical:UbuntuServer:14.04.0-LTS')
inst_1 = azure.launch('Canonical:UbuntuServer:18.04-DAILY-LTS')
```

If further customization of an instance is required, a user can pass additional arguments to the launch command and have them passed on.

```
inst = azure.launch(
    image_id='Canonical:UbuntuServer:14.04.0-LTS',
    user_data='#cloud-config\nfinal_message: "system up!"',
)
```

By default, the launch method will wait for cloud-init to finish initializing before completing. When launching multiple instances a user may not wish to wait for each instance to come up by passing the `wait=False` option.

```
instances = []
for inst in range(num_instances):
    instances.append(
        azure.launch('Canonical:UbuntuServer:18.04-DAILY-LTS', wait=False))

for instance in instances:
    instance.wait()
```

Similarly, when deleting an instance, the default action will wait for the instance to complete termination. Otherwise, the `wait=False` option can be used to start the termination of a number of instances:

```
inst.delete()

for instance in instances:
    instance.delete(wait=False)
```

An existing instance can get used by providing an instance-id.

```
instance = azure.get_instance('my-azure-vm')
```

5.1.5 Snapshots

A snapshot of an instance is used to generate a new backing Azure image. The generated image can in turn get used to launch new instances. This allows for customization of an image and then re-use of that image.

```
inst = azure.launch('Canonical:UbuntuServer:14.04.0-LTS')
inst.execute('touch /etc/foobar')
image_id_snapshot = azure.snapshot(inst)
inst_prime = azure.launch(image_id_snapshot)
```

The snapshot function returns a string of the created AMI ID.

To delete the image when the snapshot is no longer required:

```
azure.image_delete(image_id_snapshot)
```

5.2 EC2

The following page documents the AWS EC2 cloud integration in pycloudlib.

5.2.1 Credentials

To access EC2 requires users to have an access key id and secret access key. These should be set in pycloudlib.toml.

AWS Dotfile (Deprecated)

The AWS CLI, Python library boto3, and other AWS tools maintain credentials and configuration settings in a local dotfile found under the aws dotfile directory (i.e. `/home/$USER/.aws/`). If these files exist they will be used to provide login and region information.

These configuration files are normally generated when running `aws configure`:

```
$ cat /home/$USER/.aws/credentials
[default]
aws_access_key_id = <KEY_VALUE>
aws_secret_access_key = <KEY_VALUE>
$ cat /home/$USER/.aws/config
[default]
output = json
region = us-west-2
```

Passed Directly (Deprecated)

The credential and region information can also be provided directly when initializing the EC2 object:

```
ec2 = pycloudlib.EC2(
    access_key_id='KEY_VALUE',
    secret_access_key='KEY_VALUE',
    region='us-west-2'
)
```

This way different credentials or regions can be used by different objects allowing for interactions with multiple regions at the same time.

5.2.2 SSH Keys

EC2 requires an SSH key to be uploaded before using it. See the SSH Key page for more details.

5.2.3 Image Lookup

To find latest daily AMI ID for a release of Ubuntu:

```
ec2.daily_image('xenial')
'ami-537e9a30'
```

The return AMI ID can then be used for launching instances.

5.2.4 Instances

Launching an instance requires at a minimum an AMI ID. Optionally, a user can specify an instance type or a Virtual Private Cloud (VPC):

```
inst_0 = ec2.launch('ami-537e9a30')
inst_1 = ec2.launch('ami-537e9a30', instance_type='i3.metal', user_data=data)
vpc = ec2.get_or_create_vpc('private_vpc')
inst_2 = ec2.launch('ami-537e9a30', vpc=vpc)
```

If no VPC is specified the region's default VPC, including security group is used. See the Virtual Private Cloud (VPC) section below for more details on creating a custom VPC.

If further customization of an instance is required, a user can pass additional arguments to the launch command and have them passed on.

```
inst = ec2.launch(
    'ami-537e9a30',
    UserData='#cloud-config\nfinal_message: "system up!"',
    Placement={
        'AvailabilityZone': 'us-west-2a'
    },
    SecurityGroupsIds=[
        'sg-1e838479',
        'sg-e6ef7d80'
    ]
)
```

By default, the launch method will wait for cloud-init to finish initializing before completing. When launching multiple instances a user may not wish to wait for each instance to come up by passing the `wait=False` option.

```
instances = []
for inst in range(num_instances):
    instances.append(ec2.launch('ami-537e9a30', wait=False))

for instance in instances:
    instance.wait()
```

Similarly, when deleting an instance, the default action will wait for the instance to complete termination. Otherwise, the `wait=False` option can be used to start the termination of a number of instances:

```
inst.delete()

for instance in instances:
    instance.delete(wait=False)

for instance in instances:
    instance.wait_for_delete()
```

An existing instance can get used by providing an instance-id.

```
instance = ec2.get_instance('i-025795d8e55b055da')
```

5.2.5 Snapshots

A snapshot of an instance is used to generate a new backing AMI image. The generated image can in turn get used to launch new instances. This allows for customization of an image and then re-use of that image.

```
inst = ec2.launch('ami-537e9a30')
inst.update()
inst.execute('touch /etc/foobar')
snapshot = ec2.snapshot(instance.id)
inst_prime = ec2.launch(snapshot)
```

The snapshot function returns a string of the created AMI ID.

To delete the image when the snapshot is no longer required:

```
ec2.image_delete(snapshot)
```

5.2.6 Unique Operations

The following are unique operations to the EC2 cloud.

Virtual Private Clouds

If a custom VPC is required for any reason, then one can be created and then later used during instance creation.

```
vpc = ec2.get_or_create_vpc(name, ipv4_cidr='192.168.1.0/20')
ec2.launch('ami-537e9a30', vpc=vpc)
```

If the VPC is destroyed, all instances will be deleted as well.

```
vpc.delete()
```

Hot Add Storage Volumes

An instance is capable of getting additional storage hot added to it:

```
inst.add_volume(size=8, drive_type='gp2')
```

Volumes are attempted to be added at the next available location from `/dev/sd[f-z]`. However, NVMe devices will still be placed under `/dev/nvme#`.

Additional storage devices that were added will be deleted when the instance is removed.

Hot Add Network Devices

It is possible to hot add network devices to an instance.

```
inst.add_network_interface()
```

The instance will take the next available index. It is up to the user to configure the network devices once added.

Additional network devices that were added will be deleted when the instance is removed.

5.3 GCE

The following page documents the Google Cloud Engine (GCE) integration in pycloudlib.

5.3.1 Credentials

Service Account

The preferred method of connecting to GCE is to use service account credentials. See the GCE Authentication Getting Started page for more information on creating one.

Once a service account is created, generate a key file and download it to your system. Specify the credential file in pycloudlib.toml.

Export the Credentials File (deprecated)

Export the credential file as a shell variable and the Google API will automatically read the environmental variable and discover the credentials:

```
export GOOGLE_APPLICATION_CREDENTIALS="[path to keyfile.json]"
```

End User (Deprecated)

A secondary method of GCE access is to use end user credentials directly. This is not the recommended method and Google will warn the user and suggest using a service account instead.

If you do wish to continue using end user credentials, then the first step is to install the Google's Cloud SDK. On Ubuntu, this can be installed quickly as a snap with the following:

```
sudo snap install google-cloud-sdk --classic
```

Next, is to authorize the system by getting a token. This command will launch a web-browser, have you login to you Google account, and accept any agreements:

```
gcloud auth application-default login
```

The Google API will automatically check first for the above environmental variable for a service account credential and fallback to this gcloud login as a secondary option.

5.3.2 SSH Keys

GCE does not require any special key configuration. See the SSH Key page for more details.

5.3.3 Image Lookup

To find latest daily image for a release of Ubuntu:

```
gce.daily_image('bionic')
'ubuntu-1804-bionic-v20180823'
```

The return ID can then be used for launching instances.

5.3.4 Instances

The only supported function at this time is launching an instance. No other actions, including deleting the instance are supported.

5.4 IBM

The following page documents the IBM VPC cloud integration in pycloudlib.

5.4.1 Credentials

To operate on IBM VPC an IBM Cloud API key is required. This should be set in pycloudlib.toml or passed to pycloudlib.IBM at initialization time.

5.4.2 SSH Keys

IBM VPC requires an SSH key to be uploaded before using it. See the SSH Key page for more details.

5.4.3 Image Lookup

Note: IBM does not contain daily Ubuntu images.

To find latest released image ID for a release of Ubuntu:

```
ibm.released_image('xenial')
'r010-7334d328-7a1f-47d4-8dda-013e857a1f2b'
```

The return image ID can then be used for launching instances.

5.4.4 Instances

Launching an instance requires at a minimum an image ID. Optionally, a user can specify an instance type or a Virtual Private Cloud (VPC):

```
inst_0 = ibm.launch('r010-7334d328-7a1f-47d4-8dda-013e857a1f2b')
inst_1 = ibm.launch('r010-7334d328-7a1f-47d4-8dda-013e857a1f2b', instance_type='bx2-
↪metal-96x384', user_data=data)
vpc = ibm.get_or_create_vpc('custom_vpc')
inst_2 = ibm.launch('r010-7334d328-7a1f-47d4-8dda-013e857a1f2b', vpc=vpc)
```

If no VPC is specified the region's default VPC, including security group is used. See the Virtual Private Cloud (VPC) section below for more details on creating a custom VPC.

If further customization of an instance is required, a user can pass additional arguments to the launch command and have them passed on.

```
inst = ibm.launch(
    'r010-7334d328-7a1f-47d4-8dda-013e857a1f2b',
    **kwargs,
)
```

By default, the launch method will wait for cloud-init to finish initializing before completing. When launching multiple instances a user may not wish to wait for each instance to come up by passing the `wait=False` option.

```
instances = []
for inst in range(num_instances):
    instances.append(ibm.launch('r010-7334d328-7a1f-47d4-8dda-013e857a1f2b',␣
↪wait=False))

for instance in instances:
    instance.wait()
```

Similarly, when deleting an instance, the default action will wait for the instance to complete termination. Otherwise, the `wait=False` option can be used to start the termination of a number of instances:

```
inst.delete()

for instance in instances:
    instance.delete(wait=False)

for instance in instances:
    instance.wait_for_delete()
```

An existing instance can get used by providing an instance-id.

```
instance = ibm.get_instance('i-025795d8e55b055da')
```

5.4.5 Snapshots

A snapshot of an instance is used to generate a new backing Custom Image. The generated image can in turn get used to launch new instances. This allows for customization of an image and then re-use of that image.

```
inst = ibm.launch('r010-7334d328-7a1f-47d4-8dda-013e857a1f2b')
inst.update()
inst.execute('touch /etc/foobar')
snapshot = ibm.snapshot(instance.id)
inst_prime = ibm.launch(snapshot)
```

The snapshot function returns a string of the created Custom Image ID.

To delete the image when the snapshot is no longer required:

```
ibm.image_delete(snapshot)
```

5.4.6 Unique Operations

The following are unique operations to the IBM cloud.

Virtual Private Clouds

A pre-existent VPC can be set in the config file or be passed as argument to the cloud.IBM constructor. If not set, pycloudlib will default to `{region}-default-vpc`.

```
ibm = IBM(vpc="my-custom-vpc", ...)
```

Another possibility is to create a custom VPC on the fly, then one can be created and then later used during instance creation.

```
vpc = ibm.get_or_create_vpc(name)
ibm.launch('r010-7334d328-7a1f-47d4-8dda-013e857a1f2b', vpc=vpc)
```

If the VPC is destroyed, all instances and subnets will be deleted as well.

```
vpc.delete()
```

5.5 LXD

The following page documents the LXD cloud integration in pycloudlib.

5.5.1 Launching Instances

Launching instances with LXD only requires an instance name and a release name by default.

```
lxd.launch('my-instance', 'bionic')
```

Instances can be initialized or launched. The difference is initializing involves getting the required image and setting up the instance, but not starting it. The following is the same as the above command.

```
inst = lxd.init('my-instance', 'bionic')
inst.start()
```

Launch Options

Instances can take a large number of settings and options. Consult the API for a full list, however here are a few examples showing different image remotes, ephemeral instance creation, and custom settings.

```
lxd.launch(
    'pycloudlib-ephemeral', 'bionic', image_remote='ubuntu', ephemeral=True
)

lxd.launch(
    'pycloudlib-custom-hw', 'ubuntu/xenial', image_remote='images',
    network='lxdbr0', storage='default', inst_type='t2.micro', wait=False
)
```

5.5.2 Snapshots

Snapshots allow for saving and reverting to a particular point in time.

```
instance.snapshot(snapshot_name)
instance.restore(snapshot_name)
```

Snapshots can at as a base for creating new instances at a pre-configured state. See the cloning section below.

5.5.3 Cloning

Cloning instances allows for copying an existing instance or snapshot of an instance to a new container. This is useful when wanting to setup a instance with a particular state and then re-use that state over and over to avoid needing to repeat the steps to get to the initial state.

```
lxd.launch_snapshot('instance', new_instance_name)
lxd.launch_snapshot('instance\snapshot', new_instance_name)
```

5.5.4 Unique Operations

Enable KVM

Enabling KVM to work properly inside a container requires passing the `/dev/kvm` device to the container. This can be done by creating a profile and then using that profile when launching instances.

```
lxc profile create kvm
```

Add the `/dev/kvm` device to the profile.

```
devices:
  kvm:
    path: /dev/kvm
    type: unix-char
```

Then launch the instance using the default and the KVM profiles.

```
lxd.launch(
    'pycloudlib-kvm', RELEASE, profile_list=['default', 'kvm']
)
```

Nested instances

To enable nested instances of LXD containers requires making the container a privileged containers. This can be achieved by setting the appropriate configuration options.

```
lxd.launch(
    'pycloudlib-privileged',
    'bionic,
    config_dict={
        'security.nesting': 'true',
        'security.privileged': 'true'
    }
)
```

5.6 OCI

5.6.1 Credentials

Easy way

Run:

```
$ pip install oci-cli
$ oci setup config
```

When prompted:

```
location for your config: use default
user OCID: enter your user id found on the Oracle console at Identity>>Users>>User␣
↪Details
tenancy OCID: enter your tenancy id found on the Oracle cnosole at Administration>>
↪Tenancy Details
region: Choose something sensible
API Signing RSA key pair: use defaults for all prompts
* Note this ISN'T an SSH key pair
Follow instructions in your terminal for uploading your generated key
```

Now specify your `config_path` in pycloudlib.toml.

Hard way

Construct your config file manually by filling in the appropriate entries documented here: https://docs.cloud.oracle.com/en-us/iaas/Content/API/Concepts/sdkconfig.htm

Compartment id

In addition to the OCI config, pycloudlib.toml also requires you provide the compartment id. This can be found in the OCI console from the menu at Identity>Compartments>

5.6.2 SSH Keys

OCI does not require any special key configuration. See the SSH Key page for more details

5.6.3 Image Lookup

OCI doesn't have a concept of releases vs daily images, so both API calls refer to the same thing. To get the list for a release of Ubuntu:

```
oci.released_image('focal')
'ocid1.compartment.oc1..aaaaaaaanz4b63fdemmuag77dg2pi22xfyhrpq46hcgdd3dozkvqfzwwjwxa'
```

The returned image id can then be used for launching instances.

5.6.4 Instances

Launching instances requires at minimum an image_id, though instance_type (shape in Oracle terms) can also be specified, in addition to the other parameters specified by the base API.

5.6.5 Snapshots

A snapshot of an instance is used to generate a new backing image. The generated image can in turn get used to launch new instances. This allows for customization of an image and then re-use of that image.

```
inst = oci.launch(image_id)
inst.execute('touch /etc/foobar')
snapshot = oci.snapshot(instance.id)
inst_prime = oci.launch(snapshot)
```

5.7 Openstack

5.7.1 Credentials

No connection information is directly passed to pycloudlib but rather relies on **clouds.yaml** or **OS_** environment variables. See the openstack configuration docs for more information.

5.7.2 SSH Keys

Openstack can't launch instances unless an openstack managed keypair already exists. Since pycloudlib also manages keys, pycloudlib will attempt to use or create an openstack ssh keypair based on the pycloudlib keypair. If a key is provided to pycloudlib with the same name and public key that already exists in openstack, that key will be used. If no key information is provided, an openstack keypair will be created with the current user's username and public key.

5.7.3 Image ID

The image id to use for a launch must be manually passed to pycloudlib rather than determined from release name. Given that each openstack deployment can have a different setup of images, it's not practical given the information we have to guess which image to use for any particular launch.

5.7.4 Network ID

Network ID must be specified in pycloudlib.toml. Since there can be multiple networks and no concept of a default network, we can't choose which network to create an instance on.

5.7.5 Floating IPs

A floating IP is allocated and used per instance created. The IP is then deleted when the instance is deleted.

5.8 EC2

```
#!/usr/bin/env python3
# This file is part of pycloudlib. See LICENSE file for license information.
"""Basic examples of various lifecycle with an EC2 instance."""

import logging
import os

import pycloudlib
from pycloudlib.cloud import ImageType


def hot_add(ec2, daily):
    """Hot add to an instance.

    Give an example of hot adding a pair of network interfaces and a
    couple storage volumes of various sizes.
    """
    instance = ec2.launch(daily, instance_type="m4.xlarge")

    instance.add_network_interface()
    instance.add_network_interface()

    instance.add_volume(size=9)
    instance.add_volume(size=10, drive_type="gp2")

    instance.delete()

```

(continues on next page)

(continued from previous page)

```
28
29  def launch_multiple(ec2, daily):
30      """Launch multiple instances.
31
32      How to quickly launch multiple instances with EC2. This prevents
33      waiting for the instance to start each time.
34      """
35      instances = []
36      for _ in range(3):
37          instances.append(ec2.launch(daily, wait=False))
38
39      for instance in instances:
40          instance.wait()
41
42      for instance in instances:
43          instance.delete(wait=False)
44
45      for instance in instances:
46          instance.wait_for_delete()
47
48
49  def snapshot(ec2, daily):
50      """Create a snapshot from a customized image and launch it."""
51      instance = ec2.launch(daily)
52      instance.execute("touch custom_config_file")
53
54      image = ec2.snapshot(instance)
55      new_instance = ec2.launch(image)
56      new_instance.execute("ls")
57
58      new_instance.delete()
59      ec2.delete_image(image)
60      instance.delete()
61
62
63  def custom_vpc(ec2, daily):
64      """Launch instances using a custom VPC."""
65      vpc = ec2.get_or_create_vpc(name="test-vpc")
66      ec2.launch(daily, vpc=vpc)
67
68      # vpc.delete will also delete any associated instances in that VPC
69      vpc.delete()
70
71
72  def launch_basic(ec2, daily):
73      """Show basic functionality on instances.
74
75      Simple launching of an instance, run a command, and delete.
76      """
77      instance = ec2.launch(daily)
78      instance.console_log()
79      print(instance.execute("lsb_release -a"))
80
81      instance.shutdown()
82      instance.start()
83      instance.restart()
84
```

(continues on next page)

(continued from previous page)

```
    # Various Attributes
    print(instance.ip)
    print(instance.id)
    print(instance.image_id)
    print(instance.availability_zone)

    instance.delete()


def launch_pro(ec2, daily):
    """Show basic functionality on PRO instances."""
    print("Launching Pro instance...")
    instance = ec2.launch(daily)
    print(instance.execute("sudo ua status --wait"))
    print("Deleting Pro instance...")
    instance.delete()


def launch_pro_fips(ec2, daily):
    """Show basic functionality on PRO instances."""
    print("Launching Pro FIPS instance...")
    instance = ec2.launch(daily)
    print(instance.execute("sudo ua status --wait"))
    print("Deleting Pro FIPS instance...")
    instance.delete()


def handle_ssh_key(ec2, key_name):
    """Manage ssh keys to be used in the instances."""
    if key_name in ec2.list_keys():
        ec2.delete_key(key_name)

    key_pair = ec2.client.create_key_pair(KeyName=key_name)
    private_key_path = "ec2-test.pem"
    with open(private_key_path, "w", encoding="utf-8") as stream:
        stream.write(key_pair["KeyMaterial"])
    os.chmod(private_key_path, 0o600)

    # Since we are using a pem file, we don't have distinct public and
    # private key paths
    ec2.use_key(
        public_key_path=private_key_path,
        private_key_path=private_key_path,
        name=key_name,
    )


def demo():
    """Show example of using the EC2 library.

    Connects to EC2 and finds the latest daily image. Then runs
    through a number of examples.
    """
    ec2 = pycloudlib.EC2(tag="examples")
    key_name = "test-ec2"
    handle_ssh_key(ec2, key_name)

```

(continues on next page)

(continued from previous page)

```
    daily = ec2.daily_image(release="bionic")
    daily_pro = ec2.daily_image(release="bionic", image_type=ImageType.PRO)
    daily_pro_fips = ec2.daily_image(
        release="bionic", image_type=ImageType.PRO_FIPS
    )

    launch_basic(ec2, daily)
    launch_pro(ec2, daily_pro)
    launch_pro_fips(ec2, daily_pro_fips)
    custom_vpc(ec2, daily)
    snapshot(ec2, daily)
    launch_multiple(ec2, daily)
    hot_add(ec2, daily)


if __name__ == "__main__":
    logging.basicConfig(level=logging.DEBUG)
    demo()
```

5.9 GCE

```
#!/usr/bin/env python3
# This file is part of pycloudlib. See LICENSE file for license information.
"""Basic examples of various lifecycle with an GCE instance."""

import logging
import os

import pycloudlib
from pycloudlib.cloud import ImageType


def manage_ssh_key(gce):
    """Manage ssh keys for gce instances."""
    pub_key_path = "gce-pubkey"
    priv_key_path = "gce-privkey"
    pub_key, priv_key = gce.create_key_pair()

    with open(pub_key_path, "w", encoding="utf-8") as f:
        f.write(pub_key)

    with open(priv_key_path, "w", encoding="utf-8") as f:
        f.write(priv_key)

    os.chmod(pub_key_path, 0o600)
    os.chmod(priv_key_path, 0o600)

    gce.use_key(public_key_path=pub_key_path, private_key_path=priv_key_path)


def generic(gce):
    """Show example of using the GCE library.

    Connects to GCE and finds the latest daily image. Then runs
```

(continues on next page)

(continued from previous page)

```
    through a number of examples.
    """
    daily = gce.daily_image("bionic", arch="x86_64")
    inst = gce.launch(daily)
    print(inst.execute("lsb_release -a"))
    inst.delete()


def pro(gce):
    """Show example of running a GCE PRO machine."""
    daily = gce.daily_image("bionic", image_type=ImageType.PRO)
    inst = gce.launch(daily)
    print(inst.execute("sudo ua status --wait"))
    inst.delete()


def pro_fips(gce):
    """Show example of running a GCE PRO FIPS machine."""
    daily = gce.daily_image("bionic", image_type=ImageType.PRO_FIPS)
    inst = gce.launch(daily)
    print(inst.execute("sudo ua status --wait"))
    inst.delete()


def demo():
    """Show examples of launching GCP instances."""
    logging.basicConfig(level=logging.DEBUG)
    gce = pycloudlib.GCE(tag="examples")
    manage_ssh_key(gce)

    generic(gce)
    pro(gce)
    pro_fips(gce)


if __name__ == "__main__":
    demo()
```

5.10 IBM

```
#!/usr/bin/env python3
# This file is part of pycloudlib. See LICENSE file for license information.
"""Basic examples of various lifecycle with an IBM instance."""

import logging
import os

import pycloudlib


def snapshot(ibm, daily):
    """Create a snapshot from a customized image and launch it."""
    instance = ibm.launch(daily)
    instance.execute("touch custom_config_file")
```

(continues on next page)

(continued from previous page)

```
    image = ibm.snapshot(instance)
    new_instance = ibm.launch(image, name="example-snapshot")
    new_instance.execute("ls")

    new_instance.delete()
    ibm.delete_image(image)
    instance.delete()


def custom_vpc(ibm, daily):
    """Launch instances using a custom VPC."""
    vpc = ibm.get_or_create_vpc(name="test-vpc")
    ibm.launch(daily, vpc=vpc)

    # vpc.delete will also delete any associated instances in that VPC
    vpc.delete()


def launch_basic(ibm, daily, instance_type):
    """Show basic functionality on instances.

    Simple launching of an instance, run a command, and delete.
    """
    instance = ibm.launch(daily, instance_type=instance_type)
    print(instance.execute("lsb_release -a"))

    instance.shutdown()
    instance.start()
    instance.restart()

    # Various Attributes
    print(instance.ip)
    print(instance.id)

    instance.delete()


def manage_ssh_key(ibm, key_name):
    """Manage ssh keys for ibm instances."""
    if key_name in ibm.list_keys():
        ibm.delete_key(key_name)

    pub_key_path = "ibm-pubkey"
    priv_key_path = "ibm-privkey"
    pub_key, priv_key = ibm.create_key_pair()

    with open(pub_key_path, "w", encoding="utf-8") as f:
        f.write(pub_key)

    with open(priv_key_path, "w", encoding="utf-8") as f:
        f.write(priv_key)

    os.chmod(pub_key_path, 0o600)
    os.chmod(priv_key_path, 0o600)

    ibm.use_key(
```

(continues on next page)

(continued from previous page)

```
        public_key_path=pub_key_path,
        private_key_path=priv_key_path,
        name=key_name,
    )


def demo():
    """Show example of using the IBM library.

    Connects to IBM and finds the latest daily image. Then runs
    through a number of examples.
    """
    ibm = pycloudlib.IBM(tag="examples")
    manage_ssh_key(ibm, "test-ibm")

    daily = ibm.daily_image(release="bionic")

    # "bx2-metal-96x384" for a bare-metal instance
    launch_basic(ibm, daily, "bx2-2x8")
    custom_vpc(ibm, daily)
    snapshot(ibm, daily)


if __name__ == "__main__":
    logging.basicConfig(level=logging.DEBUG)
    demo()
```

5.11 LXD

```
#!/usr/bin/env python3
# This file is part of pycloudlib. See LICENSE file for license information.
"""Basic examples of various lifecycle with a LXD instance."""
import logging
import textwrap

import pycloudlib

RELEASE = "bionic"


def snapshot_instance():
    """Demonstrate snapshot functionality.

    This shows the lifecycle of booting an instance and cleaning it
    before creating a snapshot.

    Next, both create the snapshot and immediately restore the original
    instance to the snapshot level.
    Finally, launch another instance from the snapshot of the instance.
    """
    lxd = pycloudlib.LXDContainer("example-snapshot")
    inst = lxd.launch(name="pycloudlib-snapshot-base", image_id=RELEASE)

    snapshot_name = "snapshot"
```

(continues on next page)

(continued from previous page)

```
    inst.local_snapshot(snapshot_name)
    inst.restore(snapshot_name)

    child = lxd.clone(
        "%s/%s" % (inst.name, snapshot_name), "pycloudlib-snapshot-child"
    )

    child.delete()
    inst.delete_snapshot(snapshot_name)
    inst.delete(wait=False)


def image_snapshot_instance(ephemeral_instance=False):
    """Demonstrate image snapshot functionality.

    Create an snapshot image from a running instance an show
    how to launch a new instance based of this image snapshot
    """
    lxd = pycloudlib.LXDContainer("example-image-snapshot")
    inst = lxd.launch(
        name="pycloudlib-snapshot-base",
        image_id=RELEASE,
        ephemeral=ephemeral_instance,
    )
    inst.execute("touch snapshot-test.txt")
    print("Base instance output: {}".format(inst.execute("ls")))
    snapshot_image = lxd.snapshot(instance=inst)

    snapshot_inst = lxd.launch(
        name="pycloudlib-snapshot-image",
        image_id=snapshot_image,
        ephemeral=ephemeral_instance,
    )
    print("Snapshot instance output: {}".format(snapshot_inst.execute("ls")))

    snapshot_inst.delete()
    inst.delete()


def modify_instance():
    """Demonstrate how to modify and interact with an instance.

    The inits an instance and before starting it, edits the the
    container configuration.

    Once started the instance demonstrates some interactions with the
    instance.
    """
    lxd = pycloudlib.LXDContainer("example-modify")

    inst = lxd.init("pycloudlib-modify-inst", RELEASE)
    inst.edit("limits.memory", "3GB")
    inst.start()

    inst.execute("uptime > /tmp/uptime")
    inst.pull_file("/tmp/uptime", "/tmp/pulled_file")
    inst.push_file("/tmp/pulled_file", "/tmp/uptime_2")
```

(continues on next page)

(continued from previous page)

```
    inst.execute("cat /tmp/uptime_2")

    inst.delete(wait=False)


def launch_multiple():
    """Launch multiple instances.

    How to quickly launch multiple instances with LXD. This prevents
    waiting for the instance to start each time. Note that the
    wait_for_delete method is not used, as LXD does not do any waiting.
    """
    lxd = pycloudlib.LXDContainer("example-multiple")

    instances = []
    for num in range(3):
        inst = lxd.launch(
            name="pycloudlib-%s" % num, image_id=RELEASE, wait=False
        )
        instances.append(inst)

    for instance in instances:
        instance.wait()

    for instance in instances:
        instance.delete()


def launch_options():
    """Demonstrate various launching scenarios.

    First up is launching with a different profile, in this case with
    two profiles.

    Next, is launching an ephemeral instance with a different image
    remote server.

    Then, an instance with custom network, storage, and type settings.
    This is an example of booting an instance without cloud-init so
    wait is set to False.

    Finally, an instance with custom configurations options.
    """
    lxd = pycloudlib.LXDContainer("example-launch")
    kvm_profile = textwrap.dedent(
        """\
        devices:
          kvm:
            path: /dev/kvm
            type: unix-char
        """
    )

    lxd.create_profile(profile_name="kvm", profile_config=kvm_profile)

    lxd.launch(
        name="pycloudlib-kvm",
```

(continues on next page)

(continued from previous page)

```
        image_id=RELEASE,
        profile_list=["default", "kvm"],
    )
    lxd.delete_instance("pycloudlib-kvm")

    lxd.launch(
        name="pycloudlib-ephemeral",
        image_id="ubuntu:%s" % RELEASE,
        ephemeral=True,
    )
    lxd.delete_instance("pycloudlib-ephemeral")

    lxd.launch(
        name="pycloudlib-custom-hw",
        image_id="images:ubuntu/xenial",
        network="lxdbr0",
        storage="default",
        inst_type="t2.micro",
        wait=False,
    )
    lxd.delete_instance("pycloudlib-custom-hw")

    lxd.launch(
        name="pycloudlib-privileged",
        image_id=RELEASE,
        config_dict={
            "security.nesting": "true",
            "security.privileged": "true",
        },
    )
    lxd.delete_instance("pycloudlib-privileged")


def basic_lifecycle():
    """Demonstrate basic set of lifecycle operations with LXD."""
    lxd = pycloudlib.LXDContainer("example-basic")
    inst = lxd.launch(image_id=RELEASE)
    inst.delete()

    name = "pycloudlib-daily"
    inst = lxd.launch(name=name, image_id=RELEASE)
    inst.console_log()

    result = inst.execute("uptime")
    print(result)
    print(result.return_code)
    print(result.ok)
    print(result.failed)
    print(bool(result))

    inst.shutdown()
    inst.start()
    inst.restart()

    # Custom attributes
    print(inst.ephemeral)
    print(inst.state)
```

(continues on next page)

(continued from previous page)

```
    inst = lxd.get_instance(name)
    inst.delete()


def launch_virtual_machine():
    """Demonstrate launching virtual machine scenario."""
    lxd = pycloudlib.LXDVirtualMachine("example-vm")

    pub_key_path = "lxd-pubkey"
    priv_key_path = "lxd-privkey"
    pub_key, priv_key = lxd.create_key_pair()

    with open(pub_key_path, "w", encoding="utf-8") as f:
        f.write(pub_key)

    with open(priv_key_path, "w", encoding="utf-8") as f:
        f.write(priv_key)

    lxd.use_key(public_key_path=pub_key_path, private_key_path=priv_key_path)

    image_id = lxd.released_image(release=RELEASE)
    image_serial = lxd.image_serial(image_id)
    print("Image serial: {}".format(image_serial))
    name = "pycloudlib-vm"
    inst = lxd.launch(name=name, image_id=image_id)
    print("Is vm: {}".format(inst.is_vm))
    result = inst.execute("lsb_release -a")
    print(result)
    print(result.return_code)
    print(result.ok)
    print(result.failed)
    print(bool(result))

    inst_2 = lxd.get_instance(name)
    print(inst_2.execute("lsb_release -a"))

    inst.shutdown()
    inst.start()
    inst.restart()
    inst.delete()


def demo():
    """Show examples of using the LXD library."""
    basic_lifecycle()
    launch_options()
    launch_multiple()
    modify_instance()
    snapshot_instance()
    image_snapshot_instance(ephemeral_instance=False)
    launch_virtual_machine()


if __name__ == "__main__":
    logging.basicConfig(level=logging.DEBUG)
    demo()
```

5.12 OCI

```python
#!/usr/bin/env python3
# This file is part of pycloudlib. See LICENSE file for license information.
"""Basic examples of various lifecycle with an OCI instance."""

import logging
import sys
from base64 import b64encode

import pycloudlib

cloud_config = """#cloud-config
runcmd:
  - echo 'hello' > /home/ubuntu/example.txt
"""


def demo(availability_domain, compartment_id):
    """Show example of using the OCI library.

    Connects to OCI and launches released image. Then runs
    through a number of examples.
    """
    client = pycloudlib.OCI(
        "oracle-test",
        availability_domain=availability_domain,
        compartment_id=compartment_id,
    )

    instance = client.launch(
        image_id=client.released_image("focal"),
        user_data=b64encode(cloud_config.encode()).decode(),
    )

    print(instance.instance_data)
    print(instance.ip)
    instance.execute("cloud-init status --wait --long")
    print(instance.execute("cat /home/ubuntu/example.txt"))

    snapshotted_image_id = client.snapshot(instance)

    instance.delete()

    new_instance = client.launch(image_id=snapshotted_image_id)
    new_instance.delete()


if __name__ == "__main__":
    logging.basicConfig(level=logging.DEBUG)
    if len(sys.argv) != 3:
        print("Usage: oci.py <availability_domain> <compartment_id>")
        sys.exit(1)
    passed_availability_domain = sys.argv[1]
    passed_compartment_id = sys.argv[2]
    demo(passed_availability_domain, passed_compartment_id)
```

5.13 Configuration

Configuration is achieved via a configuration file. At the root of the pycloudlib repo is a file named *pycloudlib.toml.template*. This file contains stubs for the credentials necessary to connect to any individual cloud. Fill in the details appropriately and copy the file to either **~/.config/pycloudlib.toml** or **/etc/pycloudlib.toml**.

Additionally, the configuration file path can be passed to the API directly or via the **PYCLOUDLIB_CONFIG** environment variable. The order pycloudlib searches for a configuration file is:

- Passed via the API
- PYCLOUDLIB_CONFIG
- ~/.config/pycloudlib.toml
- /etc/pycloudlib.toml

5.13.1 pycloudlib.toml.template

```
############### pycloudlib.toml.template ###################################
# Copy this file to ~/.config/pycloudlib.toml or /etc/pycloudlib.toml and
# fill in the values appropriately. You can also set a PYCLOUDLIB_CONFIG
# environment variable to point to the path of the config file.
#
# After you complete this file, DO NOT CHECK IT INTO VERSION CONTROL
# It you have a secret manager like lastpass, it should go there
#
# If a key is uncommented, it is required to launch an instance on that cloud.
# Commented keys aren't required, but allow further customization for
# settings in which the defaults don't work for you. If a key has a value,
# that represents the default for that cloud.
############################################################################

[azure]
# Credentials can be found with `az ad sp create-for-rbac --sdk-auth`
client_id = ""
client_secret = ""
subscription_id = ""
tenant_id = ""
# region = "centralus"
# public_key_path = "~/.ssh/id_rsa.pub"
# private_key_path = ""  # Defaults to 'public_key_path' without the '.pub'
# key_name = ""  # Defaults to your username if not set

[ec2]
# Most values can be found in ~/.aws/credentials or ~/.aws/config
access_key_id = ""  # in ~/.aws/credentials
secret_access_key = ""  # in ~/.aws/credentials
region = ""  # in ~/.aws/config
# public_key_path = "/root/id_rsa.pub"
# private_key_path = ""  # Defaults to 'public_key_path' without the '.pub'
# key_name = ""  # can be found with `aws ec2 describe-key-pairs`


[gce]
# For a user, credentials_path should be ~/.config/gcloud/application_default_
↪credentials.json
```

(continues on next page)

(continued from previous page)

```
# For a service, in the console, create a json key in the IAM service accounts page␣
↪and download
credentials_path = "~/.config/gcloud/application_default_credentials.json"
project = ""  # glcoud config get-value project
# region = "us-west2"
# zone = "a"
# service_account_email = ""
# public_key_path = "~/.ssh/id_rsa.pub"
# private_key_path = ""  # Defaults to 'public_key_path' without the '.pub'
# key_name = ""  # Defaults to your username if not set

[ibm]
# If vpc is given, then the vpc has to belong to the same resource_group specified␣
↪here.
# resource_group = "Default"  # Defaults to `Default`
# vpc = "vpc_name"  # Defaults to `{region}-default-vpc`.
# api_key = ""  # IBM Cloud API key
# region = "eu-de"
# zone = "eu-de-2"
# public_key_path = "/root/id_rsa.pub"
# private_key_path = ""  # Defaults to 'public_key_path' without the '.pub'
# key_name = ""  # Defaults to your username if not set

[oci]
config_path = "~/.oci/config"
availability_domain = ""  # Likely in ~/.oci/oci_cli_rc
compartment_id = ""  # Likely in ~/.oci/oci_cli_rc
# public_key_path = "~/.ssh/id_rsa.pub"
# private_key_path = ""  # Defaults to 'public_key_path' without the '.pub'
# key_name = ""  # Defaults to your username if not set

[openstack]
# Openstack can be configured a number of different ways, so best to defer
# to clouds.yaml or OS_ env vars.
# See https://docs.openstack.org/openstacksdk/latest/user/config/configuration.html
network = ""  # openstack network list
# public_key_path = "~/.ssh/id_rsa.pub"
# private_key_path = ""  # Defaults to 'public_key_path' without the '.pub'
# key_name = ""  # Defaults to your username if not set

[lxd]
```

5.14 SSH Key Setup

Clouds have different expectations of whether a key should be pre-loaded before launching instances or whether a key can be specified during launch. This page goes through a few different scenarios.

5.14.1 Default Behavior

The default behavior of pycloudlib is to use the user's RSA key found in `/home/$USER/.ssh/`. On clouds where the key is referenced by a name (e.g. AWS EC2), then the value of `$USER` is used:

| Item | Default Location | | --- | --- | | Public Key | `/home/$USER/.ssh/id_rsa.pub` | | Private Key | `/home/$USER/.ssh/id_rsa` | | Name | `$USER` |

If any of these values are not correct, then the user will need to specify the key to use or upload a new key. See the following sections for more information.

5.14.2 Using the Configuration File

In pycloudlib.toml, any cloud can take the optional keys `public_key_path`, `private_key_path`, and `key_name`. If specified, these values will be used for SSH.

5.14.3 Use an Uploaded Key

Ideally if the user's SSH key as started above will not work, then the user will have already uploaded the key to be used with the cloud.

To prevent needing to upload and delete a key over-and-over a user can specify a previously uploaded key by again pointing at the public key and the name the cloud uses to reference the key:

```
cloud.use_key('/tmp/id_rsa.pub', '/tmp/private', 'powersj_tmp')
'using SSH key powersj_tmp'
```

| Item | Default Location | | ———— | ———————— | | Public Key | `/tmp/id_rsa.pub` | | Private Key | `/tmp/private` | | Name | `powersj_tmp` |

5.14.4 Upload a New Key

This is not available on all clouds, only those that require a key to be uploaded.

On AWS EC2 for example, on-the-fly SSH key usage is not allowed as a key must have been previously uploaded to the cloud. As such a user can upload a key by pointing at the public key and giving it a name. The following both uploads and tells pycloudlib which key to use in one command:

```
cloud.upload_key('/tmp/id_rsa.pub', 'powersj_tmp')
'uploading SSH key powersj_tmp'
'using SSH key powersj_tmp'
```

Uploading a key with a name that already exists will fail. Hence having the user have the keys in place before running and using `use_key()` is the preferred method.

5.14.5 Deleting an Uploaded Key

This is not available on all clouds, only those that require a key to be uploaded.

Finally, to delete an uploaded key:

```
cloud.delete_key('powersj_tmp')
'deleting SSH key powersj_tmp'
```

5.15 Images

By default, images used are based on Ubuntu's daily cloud images.

pycloudlib uses simplestreams to determine the latest daily images using the appropriate images found at Ubuntu Cloud Images site.

5.15.1 Filter

The image search is filtered based on a variety of options, which vary from cloud to cloud. Here is an example for Amazon's EC2:

```
        filters = [
            'arch=%s' % arch,
            'endpoint=%s' % 'https://ec2.%s.amazonaws.com' % self.region,
            'region=%s' % self.region,
            'release=%s' % release,
            'root_store=%s' % root_store,
            'virt=hvm',
        ]
```

This allows for the root store to be configurable by the user.

5.16 Contributing

This document describes how to contribute changes to pycloudlib.

5.16.1 Get the Source

The following demonstrates how to obtain the source from Launchpad and how to create a branch to hack on.

It is assumed you have a Launchpad account and refers to your launchpad user as LP_USER throughout.

```
git clone https://git.launchpad.net/pycloudlib
cd pycloudlib
git remote add LP_USER ssh://LP_USER@git.launchpad.net/~LP_USER/pycloudlib
git push LP_USER master
git checkout -b YOUR_BRANCH
```

5.16.2 Make Changes

Development Environment

The makefile can be used to create a Python virtual environment and do local testing:

```
# Creates a python virtual environment with all requirements
make venv
. venv/bin/activate
```

Documentation

The docs directory has its own makefile that can be used to install the dependencies required for document generation.

Documentation should be written in Markdown whenever possible.

Considerations

When making changes please keep the following in mind:

- Keep pull requests limited to a single issue
- Code must be formatted to Black standards
  - Run `tox -e format` to reformat code accordingly
- Run `tox` to execute style and lint checks
- When adding new clouds please add detailed documentation under the `docs` directory and code examples under `examples`

5.16.3 Submit a Merge Request

To submit your merge request first push your branch:

```
git push -u LP_USER YOUR_BRANCH
```

Then navigate to your personal Launchpad code page:

https://code.launchpad.net/~LP_USER/pycloudlib

And do the following:

- Click on your branch and choose 'Propose for merging'
- Target branch: set to 'master'
- Enter a commit message formatted as follows:

```
topic: short description

Detailed paragraph with change information goes here. Describe why the
changes are getting made, not what as that is obvious.

Fixes LP: #1234567
```

The submitted branch will get auto-reviewed by a bot and then a developer in the pycloudlib-devs group will review of your submitted merge.

5.16.4 Do a Review

Pull the code into a local branch:

```
git checkout -b <branch-name> <LP_USER>
git pull https://git.launchpad.net/<LP_USER>/pycodestyle.git merge_request
```

Merge, re-test, and push:

```
git checkout master
git merge <branch-name>
tox
git push origin master
```

5.17 Maintainer Notes

5.17.1 Release Checklist

Run tox

```
tox
```

Update `VERSION` file with new release number

Use Semantic Versioning:

- major release is for breaking changes
- minor release for new features/functionality
- patch release for bug fixes

Some example scenarios are below

```
1.1.1 -> 1.1.2 for a bug fix
1.1.1 -> 1.2.0 for a new feature
1.1.1 -> 2.1.0 for a breaking change
```

Push to Github

```
git commit -am "Commit message"
git push
```

Submit Pull Request on Github

Use the web UI or one of the supported CLI tools

5.18 Design

The following outlines some key points from the design of the library:

5.18.1 Images

Instances are expected to use the latest daily image, unless another image is specifically requested.

cloud-init

The images are expected to have cloud-init in them to properly start. When an instance is started, or during launch, the instance is checked for the boot complete file that cloud-init produces.

5.18.2 Instances

Instances shall use consistent operation schema across the clouds. For example:

- launch
- start
- shutdown
- restart

In addition interactions with the instance are covered by a standard set of commands:

- execute
- pull_file
- push_file
- console_log

5.18.3 Exceptions

The custom pycloudlib exceptions are located in `pycloudlib.errors`. Specific clouds can implement custom exceptions, refer to *pycloudlib.<cloud>.errors*.

Exceptions from underlying libraries will be wrapped in a `pycloudlib.errors.CloudError`, some of them will be leaked directly through for the end-user.

5.18.4 Logging

Logging is set up using the standard logging module. It is up to the user to set up their logging configuration and set the appropriate level.

Logging for paramiko, used for SSH communication, is restricted to warning level and higher, otherwise the logging is far too verbose.

5.18.5 Python Support

pycloudlib currently supports Python 3.6 and above.

pycloudlib minimum supported Python version will adhere to the Python version of the oldest Ubuntu Version with Standard Support. After that Ubuntu Version reaches the End of Standard Support, we will stop testing upstream changes against the unsupported version of Python and may introduce breaking changes. This policy may change as needed.

The following table lists the Python version supported in each Ubuntu LTS release with Standard Support:

Ubuntu Version	Python version
18.04 LTS	3.6
20.04 LTS	3.8
22.04 LTS	3.10

5.19 API

5.19.1 pycloudlib

pycloudlib package

Main pycloud module __init__.

class pycloudlib.**Azure**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *client_id: Optional[str] = None*, *client_secret: Optional[str] = None*, *subscription_id: Optional[str] = None*, *tenant_id: Optional[str] = None*, *region: Optional[str] = None*)

Bases: *pycloudlib.cloud.BaseCloud*

Azure Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *client_id: Optional[str] = None*, *client_secret: Optional[str] = None*, *subscription_id: Optional[str] = None*, *tenant_id: Optional[str] = None*, *region: Optional[str] = None*)

Initialize the connection to Azure.

Azure will try to read user credentials form the /home/$USER/.azure folder. However, we can overwrite those credentials with the provided id parameters.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **client_id** – user's client id
- **client_secret** – user's client secret access key
- **subscription_id** – user's subscription id key
- **tenant_id** – user's tenant id key
- **region** – The region where the instance will be created

create_key_pair(*key_name*)

Create a pair of ssh keys.

This method creates an a pair of ssh keys in the class resource group.

Parameters **key_name** – string, The name of the ssh resource.

daily_image(*release: str*, ***, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)

Find the image info for the latest daily image for a given release.

Parameters **release** – string, Ubuntu release to look for.

Returns A string representing an Ubuntu image

delete_image(*image_id*, ***kwargs*)

Delete an image from Azure.

Parameters **image_id** – string, The id of the image to be deleted

delete_key(*key_name*)
Delete a ssh key from the class resource group.

Parameters **key_name** – string, The name of the ssh resource.

delete_resource_group()
Delete a resource group.

get_instance(*instance_id*, *search_all=False*)
Get an instance by id.

Parameters

- **instance_id** – string, The instance name to search by
- **search_all** – boolean, Flag that indicates that if we should search for the instance in the entire reach of the subsctription id. If false, we will search only in the resource group created by this instance.

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='Standard_DS1_v2'*, *user_data=None*, *wait=True*, *name=None*, *inbound_ports=None*, ***kwargs*)
Launch virtual machine on Azure.

Parameters

- **image_id** – string, Ubuntu image to use
- **user_data** – string, user-data to pass to virtual machine
- **wait** – boolean, wait for instance to come up
- **name** – string, optional name to give the vm when launching. Default results in a name of <tag>-vm
- **inbound_ports** – List of strings, optional inbound ports to enable in the instance.
- **kwargs** – dict, other named arguments to provide to virtual_machines.begin_create_or_update

Returns Azure Instance object

Raises: ValueError on invalid image_id

list_keys()
List all ssh keys in the class resource group.

released_image(*release*)
Get the released image.

Parameters **release** – string, Ubuntu release to look for

Returns string, id of latest image

snapshot(*instance*, *clean=True*, *delete_provisioned_user=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot

- **clean** – Run instance clean method before taking snapshot
- **delete_provisioned_user** – Deletes the last provisioned user
- **kwargs** – Other named arguments specific to this implementation

Returns An image id string

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing already uploaded key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key to upload
- **name** – name to reference key by

class pycloudlib.**EC2**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *access_key_id: Optional[str] = None*, *secret_access_key: Optional[str] = None*, *region: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

EC2 Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *access_key_id: Optional[str] = None*, *secret_access_key: Optional[str] = None*, *region: Optional[str] = None*)
Initialize the connection to EC2.

boto3 will read a users /home/$USER/.aws/* files if no arguments are provided here to find values.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **access_key_id** – user's access key ID
- **secret_access_key** – user's secret access key
- **region** – region to login to

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***, *arch: str = 'x86_64'*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
Find the id of the latest daily image for a particular release.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, id of latest image

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters **image_id** – string, id of the image to delete

delete_key(*name*)
Delete an uploaded key.

Parameters **name** – The key name to delete.

get_instance(*instance_id*)
Get an instance by id.

Parameters **instance_id** –

Returns An instance object to use to manipulate the instance further.

get_or_create_vpc(*name*, *ipv4_cidr='192.168.1.0/20'*)
Create a or return matching VPC.

This can be used instead of using the default VPC to create a custom VPC for usage.

Parameters

- **name** – name of the VPC
- **ipv4_cidr** – CIDR of IPV4 subnet

Returns VPC object

image_serial(*image_id*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*)
Find the image serial of a given EC2 image ID.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='t3.micro'*, *user_data=None*, *wait=True*, *vpc=None*, ***kwargs*)
Launch instance on EC2.

Parameters

- **image_id** – string, AMI ID to use default: latest Ubuntu LTS
- **instance_type** – string, instance type to launch
- **user_data** – string, user-data to pass to instance
- **wait** – boolean, wait for instance to come up
- **vpc** – optional vpc object to create instance under
- **kwargs** – other named arguments to add to instance JSON

Returns EC2 Instance object

Raises: ValueError on invalid image_id

list_keys()
List all ssh key pair names loaded on this EC2 region.

released_image(*release: str*, ***, *arch: str = 'x86_64'*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
Find the id of the latest released image for a particular release.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use
- **root_store** – string, root store to use

Returns string, id of latest image

snapshot(*instance*, *clean=True*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

upload_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing already uploaded key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key to upload
- **name** – name to reference key by

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing already uploaded key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key to upload
- **name** – name to reference key by

class pycloudlib.**GCE**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *credentials_path: Optional[str] = None*, *project: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*, *service_account_email: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

GCE Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *credentials_path: Optional[str] = None*, *project: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*, *service_account_email: Optional[str] = None*)
Initialize the connection to GCE.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **credentials_path** – path to credentials file for GCE
- **project** – GCE project
- **region** – GCE region
- **zone** – GCE zone
- **service_account_email** – service account to bind launched instances to

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***, *arch: str = 'x86_64'*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
Find the id of the latest image for a particular release.

Parameters **release** – string, Ubuntu release to look for

Returns string, path to latest daily image

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters **image_id** – string, id of the image to delete

get_instance(*instance_id*, *name=None*)
Get an instance by id.

Parameters **instance_id** – The instance ID returned upon creation

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='n1-standard-1'*, *user_data=None*, *wait=True*, ***kwargs*)
Launch instance on GCE and print the IP address.

Parameters

- **image_id** – string, image ID for instance to use
- **instance_type** – string, instance type to launch
- **user_data** – string, user-data to pass to instance
- **wait** – boolean, wait for instance to come up
- **kwargs** – other named arguments to add to instance JSON

Raises: ValueError on invalid image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
ID of the latest released image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest released image ID for the specified release.

snapshot(*instance: pycloudlib.gce.instance.GceInstance*, *clean=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot

- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.**IBM**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *resource_group: Optional[str] = None*, *vpc: Optional[str] = None*, *api_key: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

IBM Virtual Private Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *resource_group: Optional[str] = None*, *vpc: Optional[str] = None*, *api_key: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*)
Initialize the connection to IBM VPC.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – Append a timestamped suffix to the tag string.
- **config_file** – path to pycloudlib configuration file

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***kwargs*) → str
ID of the latest daily image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest daily image ID for the specified release.

delete_image(*image_id: str*, ***kwargs*)
Delete an image.

Parameters

- **image_id** – string, id of the image to delete
- ****kwargs** – dictionary of other arguments to pass to delete_image

delete_key(*name: str*)
Delete SSH key by name.

get_instance(*instance_id: str*, ***kwargs*) → pycloudlib.instance.BaseInstance
Get an instance by id.

Parameters **instance_id** –

Returns An instance object to use to manipulate the instance further.

get_or_create_vpc(*name: str*) → pycloudlib.ibm.instance.VPC
Get a VPC by name or create it if not found.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id: str*, *instance_type: str = 'bx2-2x8'*, *user_data=None*, *wait: bool = True*, ***, *name: Optional[str] = None*, *vpc: Optional[pycloudlib.ibm.instance.VPC] = None*, ***kwargs*) → pycloudlib.instance.BaseInstance
Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type of instance to create
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – wait for instance to be live
- **name** – instance name
- **vpc** – VPC to allocate the instance in. If not given, the instance
- **be allocated in the zone's default VPC.**(*will*) –
- ****kwargs** – dictionary of other arguments to pass to launch

Returns An instance object to use to manipulate the instance further.

list_keys() → List[str]
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***, *arch: str = 'amd64'*, ***kwargs*)
ID of the latest released image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest released image ID for the specified release.

resource_group_id
Resource Group ID used to create new things under.

snapshot(*instance: pycloudlib.ibm.instance.IBMInstance*, *clean: bool = True*, ***kwargs*) → str
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key

- **name** – name to reference key by

vpc
Virtual Private Cloud.

class pycloudlib.**LXD**(**args*, ***kwargs*)
Bases: *pycloudlib.lxd.cloud.LXDContainer*

Old LXD Container Cloud Class (Kept for compatibility issues).

__init__(**args*, ***kwargs*)
Run LXDContainer constructor.

clone(*base*, *new_instance_name*)
Create copy of an existing instance or snapshot.

Uses the *lxc copy* command to create a copy of an existing instance or a snapshot. To clone a snapshot then the base is *instance_name/snapshot_name* otherwise if base is only an existing instance it will clone an instance.

Parameters

- **base** – base instance or instance/snapshot
- **new_instance_name** – name of new instance

Returns The created LXD instance object

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

create_profile(*profile_name*, *profile_config*, *force=False*)
Create a lxd profile.

Create a lxd profile and populate it with the given profile config. If the profile already exists, we will not recreate it, unless the force parameter is set to True.

Parameters

- **profile_name** – Name of the profile to be created
- **profile_config** – Config to be added to the new profile
- **force** – Force the profile creation if it already exists

daily_image(*release: str*, *arch: str = 'amd64'*, ***kwargs*)
Find the LXD fingerprint of the latest daily image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

delete_image(*image_id*, ***kwargs*)
Delete the image.

Parameters **image_id** – string, LXD image fingerprint

delete_instance(*instance_name*, *wait=True*)
Delete an instance.

Parameters

- **instance_name** – instance name to delete
- **wait** – wait for delete to complete

get_instance(*instance_id*)
Get an existing instance.

Parameters **instance_id** – instance name to get

Returns The existing instance as a LXD instance object

image_serial(*image_id*)
Find the image serial of a given LXD image.

Parameters **image_id** – string, LXD image fingerprint

Returns string, serial of latest image

init(*name*, *image_id*, *ephemeral=False*, *network=None*, *storage=None*, *inst_type=None*, *profile_list=None*, *user_data=None*, *config_dict=None*, *execute_via_ssh=True*)
Init a container.

This will initialize a container, but not launch or start it. If no remote is specified pycloudlib default to daily images.

Parameters

- **name** – string, what to call the instance
- **image_id** – string, [<remote>:]<image identifier>, the image to launch
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, optional, network name to use
- **storage** – string, optional, storage name to use
- **inst_type** – string, optional, type to use
- **profile_list** – list, optional, profile(s) to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **config_dict** – dict, optional, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

launch(*image_id*, *instance_type=None*, *user_data=None*, *wait=True*, *name=None*, *ephemeral=False*, *network=None*, *storage=None*, *profile_list=None*, *config_dict=None*, *execute_via_ssh=True*, ***kwargs*)
Set up and launch a container.

This will init and start a container with the provided settings. If no remote is specified pycloudlib defaults to daily images.

Parameters

- **image_id** – string, [<remote>:]<image>, the image to launch
- **instance_type** – string, type to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – boolean, wait for instance to start
- **name** – string, what to call the instance

- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, network name to use
- **storage** – string, storage name to use
- **profile_list** – list, profile(s) to use
- **config_dict** – dict, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

Raises: ValueError on missing image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *arch='amd64'*)
Find the LXD fingerprint of the latest released image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Take a snapshot of the passed in instance for use as image.

Parameters

- **instance** (LXDInstance) – The instance to create an image from
- **clean** – Whether to call cloud-init clean before creation
- **wait** – Whether to wait until before image is created before returning
- **name** – Name of the new image
- **stateful** – Whether to use an LXD stateful snapshot

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.**LXDContainer**(**args*, ***kwargs*)
Bases: pycloudlib.lxd.cloud._BaseLXD

LXD Containers Cloud Class.

__init__(**args*, ***kwargs*)
Run LXDContainer constructor.

clone(*base*, *new_instance_name*)
Create copy of an existing instance or snapshot.

Uses the *lxc copy* command to create a copy of an existing instance or a snapshot. To clone a snapshot then the base is *instance_name/snapshot_name* otherwise if base is only an existing instance it will clone an instance.

Parameters

- **base** – base instance or instance/snapshot
- **new_instance_name** – name of new instance

Returns The created LXD instance object

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

create_profile(*profile_name*, *profile_config*, *force=False*)
Create a lxd profile.

Create a lxd profile and populate it with the given profile config. If the profile already exists, we will not recreate it, unless the force parameter is set to True.

Parameters

- **profile_name** – Name of the profile to be created
- **profile_config** – Config to be added to the new profile
- **force** – Force the profile creation if it already exists

daily_image(*release: str*, *arch: str = 'amd64'*, ***kwargs*)
Find the LXD fingerprint of the latest daily image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

delete_image(*image_id*, ***kwargs*)
Delete the image.

Parameters **image_id** – string, LXD image fingerprint

delete_instance(*instance_name*, *wait=True*)
Delete an instance.

Parameters

- **instance_name** – instance name to delete
- **wait** – wait for delete to complete

get_instance(*instance_id*)
Get an existing instance.

Parameters **instance_id** – instance name to get

Returns The existing instance as a LXD instance object

image_serial(*image_id*)
Find the image serial of a given LXD image.

Parameters **image_id** – string, LXD image fingerprint

Returns string, serial of latest image

init(*name*, *image_id*, *ephemeral=False*, *network=None*, *storage=None*, *inst_type=None*, *profile_list=None*, *user_data=None*, *config_dict=None*, *execute_via_ssh=True*)
Init a container.

This will initialize a container, but not launch or start it. If no remote is specified pycloudlib default to daily images.

Parameters

- **name** – string, what to call the instance
- **image_id** – string, [<remote>:]<image identifier>, the image to launch
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, optional, network name to use
- **storage** – string, optional, storage name to use
- **inst_type** – string, optional, type to use
- **profile_list** – list, optional, profile(s) to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **config_dict** – dict, optional, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

launch(*image_id*, *instance_type=None*, *user_data=None*, *wait=True*, *name=None*, *ephemeral=False*, *network=None*, *storage=None*, *profile_list=None*, *config_dict=None*, *execute_via_ssh=True*, ***kwargs*)
Set up and launch a container.

This will init and start a container with the provided settings. If no remote is specified pycloudlib defaults to daily images.

Parameters

- **image_id** – string, [<remote>:]<image>, the image to launch
- **instance_type** – string, type to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – boolean, wait for instance to start
- **name** – string, what to call the instance
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, network name to use
- **storage** – string, storage name to use
- **profile_list** – list, profile(s) to use
- **config_dict** – dict, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

Raises: ValueError on missing image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *arch='amd64'*)
Find the LXD fingerprint of the latest released image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Take a snapshot of the passed in instance for use as image.

Parameters

- **instance** (LXDInstance) – The instance to create an image from
- **clean** – Whether to call cloud-init clean before creation
- **wait** – Whether to wait until before image is created before returning
- **name** – Name of the new image
- **stateful** – Whether to use an LXD stateful snapshot

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.**LXDVirtualMachine**(**args*, ***kwargs*)
Bases: pycloudlib.lxd.cloud._BaseLXD

LXD Virtual Machine Cloud Class.

__init__(**args*, ***kwargs*)
Run LXDVirtualMachine constructor.

build_necessary_profiles(*image_id*)
Build necessary profiles to launch the LXD instance.

Parameters **image_id** – string, [<remote>:]<release>, the image to build profiles for

Returns A list containing the profiles created

clone(*base*, *new_instance_name*)
Create copy of an existing instance or snapshot.

Uses the *lxc copy* command to create a copy of an existing instance or a snapshot. To clone a snapshot then the base is *instance_name/snapshot_name* otherwise if base is only an existing instance it will clone an instance.

Parameters

- **base** – base instance or instance/snapshot
- **new_instance_name** – name of new instance

Returns The created LXD instance object

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

create_profile(*profile_name*, *profile_config*, *force=False*)
Create a lxd profile.

Create a lxd profile and populate it with the given profile config. If the profile already exists, we will not recreate it, unless the force parameter is set to True.

Parameters

- **profile_name** – Name of the profile to be created
- **profile_config** – Config to be added to the new profile
- **force** – Force the profile creation if it already exists

daily_image(*release: str*, *arch: str = 'amd64'*, ***kwargs*)
Find the LXD fingerprint of the latest daily image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

delete_image(*image_id*, ***kwargs*)
Delete the image.

Parameters **image_id** – string, LXD image fingerprint

delete_instance(*instance_name*, *wait=True*)
Delete an instance.

Parameters

- **instance_name** – instance name to delete
- **wait** – wait for delete to complete

get_instance(*instance_id*)
Get an existing instance.

Parameters **instance_id** – instance name to get

Returns The existing instance as a LXD instance object

image_serial(*image_id*)
Find the image serial of a given LXD image.

Parameters **image_id** – string, LXD image fingerprint

Returns string, serial of latest image

init(*name*, *image_id*, *ephemeral=False*, *network=None*, *storage=None*, *inst_type=None*, *profile_list=None*, *user_data=None*, *config_dict=None*, *execute_via_ssh=True*)
Init a container.

This will initialize a container, but not launch or start it. If no remote is specified pycloudlib default to daily images.

Parameters

- **name** – string, what to call the instance
- **image_id** – string, [<remote>:]<image identifier>, the image to launch
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, optional, network name to use
- **storage** – string, optional, storage name to use
- **inst_type** – string, optional, type to use
- **profile_list** – list, optional, profile(s) to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **config_dict** – dict, optional, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

launch(*image_id*, *instance_type=None*, *user_data=None*, *wait=True*, *name=None*, *ephemeral=False*, *network=None*, *storage=None*, *profile_list=None*, *config_dict=None*, *execute_via_ssh=True*, ***kwargs*)
Set up and launch a container.

This will init and start a container with the provided settings. If no remote is specified pycloudlib defaults to daily images.

Parameters

- **image_id** – string, [<remote>:]<image>, the image to launch
- **instance_type** – string, type to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – boolean, wait for instance to start
- **name** – string, what to call the instance
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, network name to use
- **storage** – string, storage name to use
- **profile_list** – list, profile(s) to use
- **config_dict** – dict, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

Raises: ValueError on missing image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *arch='amd64'*)
Find the LXD fingerprint of the latest released image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Take a snapshot of the passed in instance for use as image.

Parameters

- **instance** (LXDInstance) – The instance to create an image from
- **clean** – Whether to call cloud-init clean before creation
- **wait** – Whether to wait until before image is created before returning
- **name** – Name of the new image
- **stateful** – Whether to use an LXD stateful snapshot

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.**OCI**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *availability_domain: Optional[str] = None*, *compartment_id: Optional[str] = None*, *config_path: Optional[str] = None*, *config_dict: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

OCI (Oracle) cloud class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *availability_domain: Optional[str] = None*, *compartment_id: Optional[str] = None*, *config_path: Optional[str] = None*, *config_dict: Optional[str] = None*)
Initialize the connection to OCI.

OCI must be initialized on the CLI first: https://github.com/cloud-init/qa-scripts/blob/master/doc/launching-oracle.md

Parameters

- **tag** – Name of instance
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **compartment_id** – A compartment found at https://console.us-phoenix-1.oraclecloud.com/a/identity/compartments

- **availability_domain** – One of the availability domains from: 'oci iam availability-domain list'
- **config_path** – Path of OCI config file
- **config_dict** – A dictionary containing the OCI config. Overrides the values from config_path

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, *operating_system: str = 'Canonical Ubuntu'*, ***kwargs*)
Get the daily image.

OCI just has periodic builds, so "released" and "daily" don't really make sense here. Just call the same code for both.

Should be equivalent to the cli call: oci compute image list –operating-system="Canonical Ubuntu" –operating-system-version="<xx.xx>" –sort-by='TIMECREATED' –sort-order='DESC'

Parameters

- **release** – string, Ubuntu release to look for
- **operating_system** – string, Operating system to use
- ****kwargs** – dictionary of other arguments to pass to list_images

Returns string, id of latest image

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters **image_id** – string, id of the image to delete

get_instance(*instance_id*, ***kwargs*) → pycloudlib.oci.instance.OciInstance
Get an instance by id.

Parameters

- **instance_id** – ocid of the instance
- ****kwargs** – dictionary of other arguments to pass to get_instance

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='VM.Standard2.1'*, *user_data=None*, *wait=True*, ***, *retry_strategy=None*, ***kwargs*) → pycloudlib.oci.instance.OciInstance
Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type of instance to create. https://docs.cloud.oracle.com/en-us/iaas/Content/Compute/References/computeshapes.htm
- **user_data** – used by Cloud-Init to run custom scripts or provide custom Cloud-Init configuration

- **wait** – wait for instance to be live
- **retry_strategy** – a retry strategy from oci.retry module to apply for this operation
- ****kwargs** – dictionary of other arguments to pass as LaunchInstanceDetails

Returns An instance object to use to manipulate the instance further.

Raises: ValueError on invalid image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *operating_system='Canonical Ubuntu'*)
Get the released image.

OCI just has periodic builds, so "released" and "daily" don't really make sense here. Just call the same code for both

Parameters

- **release** – string, Ubuntu release to look for
- **operating_system** – string, operating system to use

Returns string, id of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot
- **name** – (Optional) Name of created image

Returns An image object

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.**Openstack**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *network: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

Openstack cloud class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *network: Optional[str] = None*)
Initialize the connection to openstack.

Requires valid pre-configured environment variables or clouds.yaml. See https://docs.openstack.org/python-openstackclient/pike/configuration/index.html

Parameters

- **tag** – Name of instance
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **network** – Name of the network to use (from openstack network list)

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***kwargs*)
Not supported for openstack.

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters **image_id** – string, id of the image to delete

get_instance(*instance_id*, ***kwargs*) → pycloudlib.openstack.instance.OpenstackInstance
Get an instance by id.

Parameters **instance_id** – ID of instance to get

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='m1.small'*, *user_data=''*, *wait=True*, ***kwargs*) → pycloudlib.openstack.instance.OpenstackInstance
Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type (flavor) of instance to create
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – wait for instance to be live
- ****kwargs** – dictionary of other arguments to pass to launch

Returns An instance object to use to manipulate the instance further.

Raises: ValueError on invalid image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***kwargs*)
Not supported for openstack.

snapshot(*instance*, *clean=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot

- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

Subpackages

pycloudlib.azure package

Main Azure module __init__.

Subpackages

pycloudlib.azure.tests package

Tests related to pycloudlib.azure module.

Submodules

pycloudlib.azure.tests.test_cloud module

Submodules

pycloudlib.azure.cloud module

Azure Cloud type.

class pycloudlib.azure.cloud.**Azure**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *client_id: Optional[str] = None*, *client_secret: Optional[str] = None*, *subscription_id: Optional[str] = None*, *tenant_id: Optional[str] = None*, *region: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

Azure Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *client_id: Optional[str] = None*, *client_secret: Optional[str] = None*, *subscription_id: Optional[str] = None*, *tenant_id: Optional[str] = None*, *region: Optional[str] = None*)
Initialize the connection to Azure.

Azure will try to read user credentials form the /home/$USER/.azure folder. However, we can overwrite those credentials with the provided id parameters.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **client_id** – user's client id
- **client_secret** – user's client secret access key
- **subscription_id** – user's subscription id key
- **tenant_id** – user's tenant id key
- **region** – The region where the instance will be created

create_key_pair(*key_name*)
Create a pair of ssh keys.

This method creates an a pair of ssh keys in the class resource group.

Parameters **key_name** – string, The name of the ssh resource.

daily_image(*release: str*, ***, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
Find the image info for the latest daily image for a given release.

Parameters **release** – string, Ubuntu release to look for.

Returns A string representing an Ubuntu image

delete_image(*image_id*, ***kwargs*)
Delete an image from Azure.

Parameters **image_id** – string, The id of the image to be deleted

delete_key(*key_name*)
Delete a ssh key from the class resource group.

Parameters **key_name** – string, The name of the ssh resource.

delete_resource_group()
Delete a resource group.

get_instance(*instance_id*, *search_all=False*)
Get an instance by id.

Parameters

- **instance_id** – string, The instance name to search by
- **search_all** – boolean, Flag that indicates that if we should search for the instance in the entire reach of the subsctription id. If false, we will search only in the resource group created by this instance.

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='Standard_DS1_v2'*, *user_data=None*, *wait=True*, *name=None*, *inbound_ports=None*, ***kwargs*)
Launch virtual machine on Azure.

Parameters

- **image_id** – string, Ubuntu image to use
- **user_data** – string, user-data to pass to virtual machine
- **wait** – boolean, wait for instance to come up
- **name** – string, optional name to give the vm when launching. Default results in a name of <tag>-vm
- **inbound_ports** – List of strings, optional inbound ports to enable in the instance.
- **kwargs** – dict, other named arguments to provide to virtual_machines.begin_create_or_update

Returns Azure Instance object

Raises: ValueError on invalid image_id

list_keys()
List all ssh keys in the class resource group.

released_image(*release*)
Get the released image.

Parameters **release** – string, Ubuntu release to look for

Returns string, id of latest image

snapshot(*instance*, *clean=True*, *delete_provisioned_user=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – Run instance clean method before taking snapshot
- **delete_provisioned_user** – Deletes the last provisioned user
- **kwargs** – Other named arguments specific to this implementation

Returns An image id string

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing already uploaded key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key to upload
- **name** – name to reference key by

pycloudlib.azure.instance module

Azure instance.

class pycloudlib.azure.instance.**AzureInstance**(*key_pair*, *client*, *instance*)
Bases: *pycloudlib.instance.BaseInstance*

Azure backed instance.

__init__(*key_pair*, *client*, *instance*)
Set up instance.

Parameters

- **key_pair** – SSH key object
- **client** – Azure compute management client
- **instance** – created azure instance object

add_network_interface() → str
Add nic to running instance.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Return the instance console log.

Raises NotImplementedError if the cloud does not support fetching the console log for this instance.

delete(*wait=True*)
Delete instance.

execute(*command*, *stdin=None*, *description=None*, *, *use_sudo=False*, *no_log=False*, **kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

generalize()
Set the OS state of the instance to generalized.

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

id
Return instance id.

image_id
Return the image_id from which this instance was created.

install (*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return instance name.

offer
Return instance sku.

pull_file (*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file (*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface (*ip_address: str*)
Remove nic from running instance.

restart (*wait=True*, ***kwargs*)
Restart an instance.

run_script (*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown (*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance shutdown

sku
Return instance sku.

start (*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete()
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

pycloudlib.azure.util module

Azure Util Functions.

pycloudlib.azure.util.**get_client**(*resource*, *config_dict: dict*)
Get azure client based on the give resource.

This method will first verify if we can get the client by using the information provided on the login account of the user machine. If the user is not logged into Azure, we will try to get the client from the ids given by the user to this class.

Parameters

- **resource** – Azure Resource, An Azure resource that we want to get a client for.
- **config_dict** – dict, Id parameters passed by the user to this class.

Returns The client for the resource passed as parameter.

pycloudlib.azure.util.**get_image_reference_params**(*image_id*)
Return the correct parameter for image reference based on image id.

Verify if the image id is associated with a current image found on Azure Marketplace or a custom image, for example, created through a snapshot process. Depending on the image id format, we can differentiate if we should create image parameters for a Marketplace image or a custom image.

Parameters **image_id** – string, Represents a image to be used when provisioning a virtual machine

Returns A dict representing the image reference parameters that will be used to provision a virtual machine

pycloudlib.azure.util.**get_plan_params**(*image_id*, *registered_image*)
Return the correct parameter for plan based on pro image id.

Parameters

- **image_id** – string, Represents a image to be used when provisioning a virtual machine
- **registered_image** – dict, Represents the base image used for creating the image referenced by image_id. This will only happen for snapshot images.

Returns A dict representing the plan parameters that will be used to provision a virtual machine

pycloudlib.azure.util.**get_resource_group_name_from_id**(*resource_id*)
Retrieve the resource group name of a resource.

Parameters **resource_id** – string, the resource id

Returns A string representing the resource group

pycloudlib.azure.util.**get_resource_name_from_id**(*resource_id*)
Retrieve the name of a resource.

Parameters **resource_id** – string, the resource id

Returns A string representing the resource name

pycloudlib.azure.util.**is_pro_image**(*image_id*, *registered_image*)
Verify if the image id represents a pro image.

Check the image id string for patterns found only on pro images. However, snapshot images do not have pro information on their image id. We are enconding that information on the registered_image dict, which represents the base image that created the snapshot. Therefore, we fail at looking in the image id string, we look it up at the registered_image dict.

Parameters

- **image_id** – string, Represents a image to be used when provisioning a virtual machine
- **registered_image** – dict, Represents the base image used for creating the image referenced by image_id. This will only happen for snapshot images.

Returns A boolean indicating if the image is pro image

pycloudlib.azure.util.**parse_image_id**(*image_id*)
Extract publisher, offer, sku and optional version from image_id.

The image_id is expected to be a string in the following format: Canonical:UbuntuServer:19.10-DAILY[:latest]

Parameters **image_id** – string, The image id

Returns Dict with publisher, offer and sku and optional version keys.

pycloudlib.ec2 package

Main EC2 module __init__.

Submodules

pycloudlib.ec2.cloud module

AWS EC2 Cloud type.

class pycloudlib.ec2.cloud.**EC2**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *access_key_id: Optional[str] = None*, *secret_access_key: Optional[str] = None*, *region: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

EC2 Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *access_key_id: Optional[str] = None*, *secret_access_key: Optional[str] = None*, *region: Optional[str] = None*)
Initialize the connection to EC2.

boto3 will read a users /home/$USER/.aws/* files if no arguments are provided here to find values.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **access_key_id** – user's access key ID
- **secret_access_key** – user's secret access key
- **region** – region to login to

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***, *arch: str = 'x86_64'*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
Find the id of the latest daily image for a particular release.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, id of latest image

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters **image_id** – string, id of the image to delete

delete_key(*name*)
Delete an uploaded key.

Parameters **name** – The key name to delete.

get_instance(*instance_id*)
Get an instance by id.

Parameters **instance_id** –

Returns An instance object to use to manipulate the instance further.

get_or_create_vpc(*name*, *ipv4_cidr='192.168.1.0/20'*)
Create a or return matching VPC.

This can be used instead of using the default VPC to create a custom VPC for usage.

Parameters

- **name** – name of the VPC
- **ipv4_cidr** – CIDR of IPV4 subnet

Returns VPC object

image_serial(*image_id*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*)
Find the image serial of a given EC2 image ID.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='t3.micro'*, *user_data=None*, *wait=True*, *vpc=None*, ***kwargs*)
Launch instance on EC2.

Parameters

- **image_id** – string, AMI ID to use default: latest Ubuntu LTS
- **instance_type** – string, instance type to launch
- **user_data** – string, user-data to pass to instance
- **wait** – boolean, wait for instance to come up
- **vpc** – optional vpc object to create instance under
- **kwargs** – other named arguments to add to instance JSON

Returns EC2 Instance object

Raises: ValueError on invalid image_id

list_keys()
List all ssh key pair names loaded on this EC2 region.

released_image(*release: str*, ***, *arch: str = 'x86_64'*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
Find the id of the latest released image for a particular release.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use
- **root_store** – string, root store to use

Returns string, id of latest image

snapshot(*instance*, *clean=True*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

upload_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing already uploaded key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key to upload
- **name** – name to reference key by

use_key (*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing already uploaded key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key to upload
- **name** – name to reference key by

pycloudlib.ec2.instance module

EC2 instance.

class pycloudlib.ec2.instance.**EC2Instance**(*key_pair*, *client*, *instance*)
Bases: *pycloudlib.instance.BaseInstance*

EC2 backed instance.

__init__(*key_pair*, *client*, *instance*)
Set up instance.

Parameters

- **key_pair** – SSH key object
- **client** – boto3 client object
- **instance** – created boto3 instance object

add_network_interface() → str
Add network interface to instance.

Creates an ENI device and attaches it to the running instance. This is effectively a hot-add of a network device. Returns the IP address of the added network interface as a string.

See the AWS documentation for more info: https://boto3.readthedocs.io/en/latest/reference/services/ec2.html?#EC2.Client.create_network_interface https://boto3.readthedocs.io/en/latest/reference/services/ec2.html?#EC2.Client.attach_network_interface

add_volume(*size=8*, *drive_type='gp2'*)
Add storage volume to instance.

Creates an EBS volume and attaches it to the running instance. This is effectively a hot-add of a storage device.

See AWS documentation for more info: https://boto3.readthedocs.io/en/latest/reference/services/ec2.html?#EC2.Client.create_volume https://boto3.readthedocs.io/en/latest/reference/services/ec2.html?#EC2.Client.attach_volume

Parameters

- **size** – Size in GB of the drive to add
- **drive_type** – Type of EBS volume to add

availability_zone
Return availability zone.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Collect console log from instance.

The console log is buffered and not always present, therefore may return empty string.

Returns The console log or error message

delete(*wait=True*)
Delete instance.

execute(*command*, *stdin=None*, *description=None*, ***, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

id
Return id of instance.

image_id
Return id of instance.

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return id of instance.

pull_file(*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file(*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface(*ip_address*)
Remove network interface based on IP address.

Find the NIC from the IP, detach from the instance, then delete the NIC.

restart(*wait=True*, ***kwargs*)
Restart an instance.

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown(*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance shutdown

start(*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete()
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

pycloudlib.ec2.util module

EC2 Util Functions.

pycloudlib.ec2.vpc module

Used to define custom Virtual Private Clouds (VPC).

class pycloudlib.ec2.vpc.**VPC**(*vpc*)
Bases: object

Virtual Private Cloud class proxy for AWS VPC resource.

__init__(*vpc*)
Create a VPC proxy instance for an AWS VPC resource.

Parameters **vpc_id** – Optional ID of existing VPC object to return

classmethod create(*resource*, *name*, *ipv4_cidr='192.168.1.0/20'*)
Create a pycloudlib.ec2.VPC proxy for an AWS VPC resource.

Parameters

- **resource** – EC2 resource client
- **name** – String for the name or tag of the VPC
- **ipv4_cidr** – String of the CIDR for IPV4 subnet to associate with the VPC.

Returns pycloudlib.ec2.VPC instance

delete()
Terminate all associated instances and delete an entire VPC.

classmethod from_existing(*resource*, *vpc_id*)
Wrap an existing boto3 EC2 VPC resource given the vpc_id.

Parameters

- **resource** – EC2 resource client
- **vpc_id** – String for an existing VPC id.

Returns pycloudlib.ec2.VPC instance

id
ID of the VPC.

name
Name of the VPC from tags.

pycloudlib.gce package

Main GCE module __init__.

Subpackages

pycloudlib.gce.tests package

Main GCE tests module __init__.

Submodules

pycloudlib.gce.tests.test_cloud module

Submodules

pycloudlib.gce.cloud module

GCE Cloud type.

This is an initial implementation of the GCE class. It enables authentication into the cloud, finding an image, and launching an instance. It however, does not allow any further actions from occurring.

class pycloudlib.gce.cloud.**GCE**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *credentials_path: Optional[str] = None*, *project: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*, *service_account_email: Optional[str] = None*)

Bases: *pycloudlib.cloud.BaseCloud*

GCE Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *credentials_path: Optional[str] = None*, *project: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*, *service_account_email: Optional[str] = None*)

Initialize the connection to GCE.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **credentials_path** – path to credentials file for GCE
- **project** – GCE project
- **region** – GCE region
- **zone** – GCE zone
- **service_account_email** – service account to bind launched instances to

create_key_pair()

Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***, *arch: str = 'x86_64'*, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)

Find the id of the latest image for a particular release.

Parameters **release** – string, Ubuntu release to look for

Returns string, path to latest daily image

delete_image(*image_id*, ***kwargs*)

Delete an image.

Parameters **image_id** – string, id of the image to delete

get_instance(*instance_id*, *name=None*)
Get an instance by id.

Parameters **instance_id** – The instance ID returned upon creation

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='n1-standard-1'*, *user_data=None*, *wait=True*, ***kwargs*)
Launch instance on GCE and print the IP address.

Parameters

- **image_id** – string, image ID for instance to use
- **instance_type** – string, instance type to launch
- **user_data** – string, user-data to pass to instance
- **wait** – boolean, wait for instance to come up
- **kwargs** – other named arguments to add to instance JSON

Raises: ValueError on invalid image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***, *image_type: pycloudlib.cloud.ImageType = <ImageType.GENERIC: 'generic'>*, ***kwargs*)
ID of the latest released image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest released image ID for the specified release.

snapshot(*instance: pycloudlib.gce.instance.GceInstance*, *clean=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

pycloudlib.gce.errors module

Module containing errors specific to gce.

exception pycloudlib.gce.errors.**GceException**
Bases: *pycloudlib.errors.PycloudlibException*

Represents an error from the GCE API.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

pycloudlib.gce.instance module

GCE instance.

class pycloudlib.gce.instance.**GceInstance**(*key_pair*, *instance_id*, *project*, *zone*, *credentials_path*, ***, *name=None*)
Bases: *pycloudlib.instance.BaseInstance*

GCE backed instance.

__init__(*key_pair*, *instance_id*, *project*, *zone*, *credentials_path*, ***, *name=None*)
Set up the instance.

Parameters

- **key_pair** – A KeyPair for SSH interactions
- **instance_id** – Id returned when creating the instance
- **project** – Project instance was created in
- **zone** – Zone instance was created in

add_network_interface() → str
Add nic to running instance.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Return the instance console log.

Raises NotImplementedError if the cloud does not support fetching the console log for this instance.

delete(*wait=True*)
Delete the instance.

Parameters **wait** – wait for instance to be deleted

execute(*command*, *stdin=None*, *description=None*, ***, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

id
Return the instance id.

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return the instance name.

pull_file(*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file(*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface(*ip_address: str*)
Remove nic from running instance.

restart(*wait=True*, ***kwargs*)
Restart an instance.

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents

- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown (*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance to shutdown

start (*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update ()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait ()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete (*sleep_seconds=30*, *raise_on_fail=False*)
Wait for instance to be deleted.

wait_for_restart (*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop ()
Wait for instance stop.

pycloudlib.gce.util module

Common GCE utils.

pycloudlib.gce.util.**get_credentials** (*credentials_path*)
Get GCE account credentials.

Try service account credentials first. If those fail, try the environment

pycloudlib.gce.util.**raise_on_error** (*response*)
Look for errors in response and raise if found.

pycloudlib.ibm package

IBM's __init__.

exception pycloudlib.ibm.**IBMException**
Bases: *pycloudlib.errors.PycloudlibException*

IBM exception root.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

class pycloudlib.ibm.**IBMInstance**(*key_pair*, *, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, *floating_ip: Optional[dict] = None*)
Bases: *pycloudlib.instance.BaseInstance*

Base instance object.

__init__(*key_pair*, *, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, *floating_ip: Optional[dict] = None*)
Set up instance.

add_network_interface() → str
Add nic to running instance.

boot_volume_id
Boot volume ID.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Return the instance console log.

Raises NotImplementedError if the cloud does not support fetching the console log for this instance.

static create_raw_instance(*client: ibm_vpc.vpc_v1.VpcV1*, *, *name: str*, *image_id: str*, *vpc: pycloudlib.ibm.instance.VPC*, *instance_type: str*, *resource_group_id: str*, *zone: str*, *user_data=None*, *key_id: str*) → dict
Create and return a raw IBM instance.

delete(*wait=True*)
Delete the instance.

Parameters **wait** – wait for instance to be deleted

execute(*command*, *stdin=None*, *description=None*, *, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

classmethod find_existing(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *instance_id: str*, ***kwargs*) → pycloudlib.ibm.instance.IBMInstance
Find an instance by ID.

classmethod from_existing(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, ***kwargs*) → pycloudlib.ibm.instance.IBMInstance
Instantiate *self* from *instance*.

If *floating_ip* is not given, it will try to discover an associated Floating Ip.

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

id
Instance ID.

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return instance name.

pull_file(*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file(*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface(*ip_address: str*)
Remove nic from running instance.

restart(*wait=True*, ***kwargs*)
Restart an instance.

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown(*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance to shutdown

start(*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete(*sleep_seconds=30*, *raise_on_fail=False*)
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

classmethod with_floating_ip(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, *floating_ip: dict*, ***kwargs*) → pycloudlib.ibm.instance.IBMInstance
Instantiate *self* from *instance* associated to *floating_ip*.

class pycloudlib.ibm.**VPC**(*key_pair*, ***, *client: ibm_vpc.vpc_v1.VpcV1*, *vpc: dict*, *resource_group_id: str*, *subnet: Optional[pycloudlib.ibm.instance._Subnet] = None*, ***_kwargs*)
Bases: object

Virtual Private Cloud class proxy for IBM VPC resource.

__init__(*key_pair*, ***, *client: ibm_vpc.vpc_v1.VpcV1*, *vpc: dict*, *resource_group_id: str*, *subnet: Optional[pycloudlib.ibm.instance._Subnet] = None*, ***_kwargs*)
Init a *VPC*.

classmethod create(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *name: str*, *resource_group_id: str*, *zone: str*, ***kwargs*) → pycloudlib.ibm.instance.VPC
Create a VPC.

Creates a Subnet and adds inbound rule to accept SSH connections to the default Security Group.

delete() → None
Delete VPC.

Note: This will delete all instances and subnets living in the VPC.

classmethod from_default(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *resource_group_id: str*, *region: str*, *zone: str*, ***kwargs*) → pycloudlib.ibm.instance.VPC
Find the *default* VPC and Subnet.

classmethod from_existing(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *name: str*, *resource_group_id: str*, *zone: str*, ***kwargs*) → pycloudlib.ibm.instance.VPC
Find a VPC by name.

Try to discover a Subnet within it or create it if not found.

id
VPC ID.

name
VPC name.

subnet_id
Subnet ID.

Subpackages

pycloudlib.ibm.tests package

IBM test package.

Submodules

pycloudlib.ibm.tests.test_util module

Submodules

pycloudlib.ibm.cloud module

IBM Cloud type.

class pycloudlib.ibm.cloud.**IBM**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *resource_group: Optional[str] = None*, *vpc: Optional[str] = None*, *api_key: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*)

Bases: *pycloudlib.cloud.BaseCloud*

IBM Virtual Private Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *resource_group: Optional[str] = None*, *vpc: Optional[str] = None*, *api_key: Optional[str] = None*, *region: Optional[str] = None*, *zone: Optional[str] = None*)

Initialize the connection to IBM VPC.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – Append a timestamped suffix to the tag string.
- **config_file** – path to pycloudlib configuration file

create_key_pair()

Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***kwargs*) → str

ID of the latest daily image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest daily image ID for the specified release.

delete_image(*image_id: str*, ***kwargs*)

Delete an image.

Parameters

- **image_id** – string, id of the image to delete
- ****kwargs** – dictionary of other arguments to pass to delete_image

delete_key(*name: str*)
Delete SSH key by name.

get_instance(*instance_id: str*, ***kwargs*) → pycloudlib.instance.BaseInstance
Get an instance by id.

Parameters **instance_id** –

Returns An instance object to use to manipulate the instance further.

get_or_create_vpc(*name: str*) → pycloudlib.ibm.instance.VPC
Get a VPC by name or create it if not found.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id: str*, *instance_type: str = 'bx2-2x8'*, *user_data=None*, *wait: bool = True*, ***, *name: Optional[str] = None*, *vpc: Optional[pycloudlib.ibm.instance.VPC] = None*, ***kwargs*) → pycloudlib.instance.BaseInstance
Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type of instance to create
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – wait for instance to be live
- **name** – instance name
- **vpc** – VPC to allocate the instance in. If not given, the instance
- **be allocated in the zone's default VPC.**(*will*) –
- ****kwargs** – dictionary of other arguments to pass to launch

Returns An instance object to use to manipulate the instance further.

list_keys() → List[str]
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***, *arch: str = 'amd64'*, ***kwargs*)
ID of the latest released image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest released image ID for the specified release.

resource_group_id
Resource Group ID used to create new things under.

snapshot(*instance: pycloudlib.ibm.instance.IBMInstance*, *clean: bool = True*, ***kwargs*) → str
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

vpc
Virtual Private Cloud.

pycloudlib.ibm.errors module

Module containing errors specific to ibm.

exception pycloudlib.ibm.errors.**IBMException**
Bases: *pycloudlib.errors.PycloudlibException*

IBM exception root.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

pycloudlib.ibm.instance module

Base class for all instances to provide consistent set of functions.

class pycloudlib.ibm.instance.**IBMInstance**(*key_pair*, ***, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, *floating_ip: Optional[dict] = None*)
Bases: *pycloudlib.instance.BaseInstance*

Base instance object.

__init__(*key_pair*, ***, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, *floating_ip: Optional[dict] = None*)
Set up instance.

add_network_interface() → str
Add nic to running instance.

boot_volume_id
Boot volume ID.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Return the instance console log.

Raises NotImplementedError if the cloud does not support fetching the console log for this instance.

static create_raw_instance(*client: ibm_vpc.vpc_v1.VpcV1*, ***, *name: str*, *image_id: str*, *vpc: pycloudlib.ibm.instance.VPC*, *instance_type: str*, *resource_group_id: str*, *zone: str*, *user_data=None*, *key_id: str*) → dict
Create and return a raw IBM instance.

delete(*wait=True*)
Delete the instance.

Parameters **wait** – wait for instance to be deleted

execute(*command*, *stdin=None*, *description=None*, ***, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

classmethod find_existing(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *instance_id: str*, ***kwargs*) → pycloudlib.ibm.instance.IBMInstance
Find an instance by ID.

classmethod from_existing(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, ***kwargs*) → pycloudlib.ibm.instance.IBMInstance
Instantiate *self* from *instance*.

If *floating_ip* is not given, it will try to discover an associated Floating Ip.

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

id
Instance ID.

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return instance name.

pull_file(*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file(*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface(*ip_address: str*)
Remove nic from running instance.

restart(*wait=True*, ***kwargs*)
Restart an instance.

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown(*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance to shutdown

start(*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete(*sleep_seconds=30*, *raise_on_fail=False*)
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

classmethod with_floating_ip(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *instance: dict*, *floating_ip: dict*, ***kwargs*) → pycloudlib.ibm.instance.IBMInstance
Instantiate *self* from *instance* associated to *floating_ip*.

class pycloudlib.ibm.instance.**VPC**(*key_pair*, ***, *client: ibm_vpc.vpc_v1.VpcV1*, *vpc: dict*, *resource_group_id: str*, *subnet: Optional[pycloudlib.ibm.instance._Subnet] = None*, ***_kwargs*)
Bases: object

Virtual Private Cloud class proxy for IBM VPC resource.

__init__(*key_pair*, ***, *client: ibm_vpc.vpc_v1.VpcV1*, *vpc: dict*, *resource_group_id: str*, *subnet: Optional[pycloudlib.ibm.instance._Subnet] = None*, ***_kwargs*)
Init a *VPC*.

classmethod create(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *name: str*, *resource_group_id: str*, *zone: str*, ***kwargs*) → pycloudlib.ibm.instance.VPC
Create a VPC.

Creates a Subnet and adds inbound rule to accept SSH connections to the default Security Group.

delete() → None
Delete VPC.

Note: This will delete all instances and subnets living in the VPC.

classmethod from_default(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *resource_group_id: str*, *region: str*, *zone: str*, ***kwargs*) → pycloudlib.ibm.instance.VPC
Find the *default* VPC and Subnet.

classmethod from_existing(**args*, *client: ibm_vpc.vpc_v1.VpcV1*, *name: str*, *resource_group_id: str*, *zone: str*, ***kwargs*) → pycloudlib.ibm.instance.VPC
Find a VPC by name.

Try to discover a Subnet within it or create it if not found.

id
VPC ID.

name
VPC name.

subnet_id
Subnet ID.

pycloudlib.lxd package

Main LXD module __init__.

Subpackages

pycloudlib.lxd.tests package

Tests related to pycloudlib.lxd module.

Submodules

pycloudlib.lxd.tests.test_cloud module

pycloudlib.lxd.tests.test_defaults module

pycloudlib.lxd.tests.test_images module

pycloudlib.lxd.tests.test_instance module

Submodules

pycloudlib.lxd.cloud module

LXD Cloud type.

class pycloudlib.lxd.cloud.**LXD**(**args*, ***kwargs*)
Bases: *pycloudlib.lxd.cloud.LXDContainer*

Old LXD Container Cloud Class (Kept for compatibility issues).

__init__(**args*, ***kwargs*)
Run LXDContainer constructor.

clone(*base*, *new_instance_name*)
Create copy of an existing instance or snapshot.

Uses the *lxc copy* command to create a copy of an existing instance or a snapshot. To clone a snapshot then the base is *instance_name/snapshot_name* otherwise if base is only an existing instance it will clone an instance.

Parameters

- **base** – base instance or instance/snapshot
- **new_instance_name** – name of new instance

Returns The created LXD instance object

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

create_profile(*profile_name*, *profile_config*, *force=False*)
Create a lxd profile.

Create a lxd profile and populate it with the given profile config. If the profile already exists, we will not recreate it, unless the force parameter is set to True.

Parameters

- **profile_name** – Name of the profile to be created
- **profile_config** – Config to be added to the new profile
- **force** – Force the profile creation if it already exists

daily_image(*release: str*, *arch: str = 'amd64'*, ***kwargs*)
Find the LXD fingerprint of the latest daily image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

delete_image(*image_id*, ***kwargs*)
Delete the image.

Parameters **image_id** – string, LXD image fingerprint

delete_instance(*instance_name*, *wait=True*)
Delete an instance.

Parameters

- **instance_name** – instance name to delete
- **wait** – wait for delete to complete

get_instance(*instance_id*)
Get an existing instance.

Parameters **instance_id** – instance name to get

Returns The existing instance as a LXD instance object

image_serial(*image_id*)
Find the image serial of a given LXD image.

Parameters **image_id** – string, LXD image fingerprint

Returns string, serial of latest image

init(*name*, *image_id*, *ephemeral=False*, *network=None*, *storage=None*, *inst_type=None*, *profile_list=None*, *user_data=None*, *config_dict=None*, *execute_via_ssh=True*)
Init a container.

This will initialize a container, but not launch or start it. If no remote is specified pycloudlib default to daily images.

Parameters

- **name** – string, what to call the instance
- **image_id** – string, [<remote>:]<image identifier>, the image to launch
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, optional, network name to use
- **storage** – string, optional, storage name to use
- **inst_type** – string, optional, type to use
- **profile_list** – list, optional, profile(s) to use
- **user_data** – used by cloud-init to run custom scripts/configuration

- **config_dict** – dict, optional, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

launch(*image_id*, *instance_type=None*, *user_data=None*, *wait=True*, *name=None*, *ephemeral=False*, *network=None*, *storage=None*, *profile_list=None*, *config_dict=None*, *execute_via_ssh=True*, ***kwargs*)

Set up and launch a container.

This will init and start a container with the provided settings. If no remote is specified pycloudlib defaults to daily images.

Parameters

- **image_id** – string, [<remote>:]<image>, the image to launch
- **instance_type** – string, type to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – boolean, wait for instance to start
- **name** – string, what to call the instance
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, network name to use
- **storage** – string, storage name to use
- **profile_list** – list, profile(s) to use
- **config_dict** – dict, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

Raises: ValueError on missing image_id

list_keys()

List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *arch='amd64'*)

Find the LXD fingerprint of the latest released image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

snapshot(*instance*, *clean=True*, *name=None*)

Take a snapshot of the passed in instance for use as image.

Parameters

- **instance** (LXDInstance) – The instance to create an image from
- **clean** – Whether to call cloud-init clean before creation

- **wait** – Whether to wait until before image is created before returning
- **name** – Name of the new image
- **stateful** – Whether to use an LXD stateful snapshot

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.lxd.cloud.**LXDContainer**(**args*, ***kwargs*)
Bases: pycloudlib.lxd.cloud._BaseLXD

LXD Containers Cloud Class.

__init__(**args*, ***kwargs*)
Run LXDContainer constructor.

clone(*base*, *new_instance_name*)
Create copy of an existing instance or snapshot.

Uses the *lxc copy* command to create a copy of an existing instance or a snapshot. To clone a snapshot then the base is *instance_name/snapshot_name* otherwise if base is only an existing instance it will clone an instance.

Parameters

- **base** – base instance or instance/snapshot
- **new_instance_name** – name of new instance

Returns The created LXD instance object

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

create_profile(*profile_name*, *profile_config*, *force=False*)
Create a lxd profile.

Create a lxd profile and populate it with the given profile config. If the profile already exists, we will not recreate it, unless the force parameter is set to True.

Parameters

- **profile_name** – Name of the profile to be created
- **profile_config** – Config to be added to the new profile
- **force** – Force the profile creation if it already exists

daily_image(*release: str*, *arch: str = 'amd64'*, ***kwargs*)
Find the LXD fingerprint of the latest daily image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

delete_image(*image_id*, ***kwargs*)
Delete the image.

Parameters **image_id** – string, LXD image fingerprint

delete_instance(*instance_name*, *wait=True*)
Delete an instance.

Parameters

- **instance_name** – instance name to delete
- **wait** – wait for delete to complete

get_instance(*instance_id*)
Get an existing instance.

Parameters **instance_id** – instance name to get

Returns The existing instance as a LXD instance object

image_serial(*image_id*)
Find the image serial of a given LXD image.

Parameters **image_id** – string, LXD image fingerprint

Returns string, serial of latest image

init(*name*, *image_id*, *ephemeral=False*, *network=None*, *storage=None*, *inst_type=None*, *profile_list=None*, *user_data=None*, *config_dict=None*, *execute_via_ssh=True*)
Init a container.

This will initialize a container, but not launch or start it. If no remote is specified pycloudlib default to daily images.

Parameters

- **name** – string, what to call the instance
- **image_id** – string, [<remote>:]<image identifier>, the image to launch
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, optional, network name to use
- **storage** – string, optional, storage name to use
- **inst_type** – string, optional, type to use
- **profile_list** – list, optional, profile(s) to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **config_dict** – dict, optional, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

launch(*image_id*, *instance_type=None*, *user_data=None*, *wait=True*, *name=None*, *ephemeral=False*, *network=None*, *storage=None*, *profile_list=None*, *config_dict=None*, *execute_via_ssh=True*, ***kwargs*)
Set up and launch a container.

This will init and start a container with the provided settings. If no remote is specified pycloudlib defaults to daily images.

Parameters

- **image_id** – string, [<remote>:]<image>, the image to launch
- **instance_type** – string, type to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – boolean, wait for instance to start
- **name** – string, what to call the instance
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, network name to use
- **storage** – string, storage name to use
- **profile_list** – list, profile(s) to use
- **config_dict** – dict, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

Raises: ValueError on missing image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *arch='amd64'*)
Find the LXD fingerprint of the latest released image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Take a snapshot of the passed in instance for use as image.

Parameters

- **instance** (LXDInstance) – The instance to create an image from
- **clean** – Whether to call cloud-init clean before creation
- **wait** – Whether to wait until before image is created before returning
- **name** – Name of the new image
- **stateful** – Whether to use an LXD stateful snapshot

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload

- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.lxd.cloud.**LXDVirtualMachine**(**args*, ***kwargs*)
Bases: pycloudlib.lxd.cloud._BaseLXD

LXD Virtual Machine Cloud Class.

__init__(**args*, ***kwargs*)
Run LXDVirtualMachine constructor.

build_necessary_profiles(*image_id*)
Build necessary profiles to launch the LXD instance.

Parameters **image_id** – string, [<remote>:]<release>, the image to build profiles for

Returns A list containing the profiles created

clone(*base*, *new_instance_name*)
Create copy of an existing instance or snapshot.

Uses the *lxc copy* command to create a copy of an existing instance or a snapshot. To clone a snapshot then the base is *instance_name/snapshot_name* otherwise if base is only an existing instance it will clone an instance.

Parameters

- **base** – base instance or instance/snapshot
- **new_instance_name** – name of new instance

Returns The created LXD instance object

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

create_profile(*profile_name*, *profile_config*, *force=False*)
Create a lxd profile.

Create a lxd profile and populate it with the given profile config. If the profile already exists, we will not recreate it, unless the force parameter is set to True.

Parameters

- **profile_name** – Name of the profile to be created
- **profile_config** – Config to be added to the new profile
- **force** – Force the profile creation if it already exists

daily_image(*release: str*, *arch: str = 'amd64'*, ***kwargs*)
Find the LXD fingerprint of the latest daily image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

delete_image(*image_id*, ***kwargs*)
Delete the image.

Parameters **image_id** – string, LXD image fingerprint

delete_instance(*instance_name*, *wait=True*)
Delete an instance.

Parameters

- **instance_name** – instance name to delete
- **wait** – wait for delete to complete

get_instance(*instance_id*)
Get an existing instance.

Parameters **instance_id** – instance name to get

Returns The existing instance as a LXD instance object

image_serial(*image_id*)
Find the image serial of a given LXD image.

Parameters **image_id** – string, LXD image fingerprint

Returns string, serial of latest image

init(*name*, *image_id*, *ephemeral=False*, *network=None*, *storage=None*, *inst_type=None*, *profile_list=None*, *user_data=None*, *config_dict=None*, *execute_via_ssh=True*)
Init a container.

This will initialize a container, but not launch or start it. If no remote is specified pycloudlib default to daily images.

Parameters

- **name** – string, what to call the instance
- **image_id** – string, [<remote>:]<image identifier>, the image to launch
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, optional, network name to use
- **storage** – string, optional, storage name to use
- **inst_type** – string, optional, type to use
- **profile_list** – list, optional, profile(s) to use
- **user_data** – used by cloud-init to run custom scripts/configuration
- **config_dict** – dict, optional, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

launch(*image_id*, *instance_type=None*, *user_data=None*, *wait=True*, *name=None*, *ephemeral=False*, *network=None*, *storage=None*, *profile_list=None*, *config_dict=None*, *execute_via_ssh=True*, ***kwargs*)
Set up and launch a container.

This will init and start a container with the provided settings. If no remote is specified pycloudlib defaults to daily images.

Parameters

- **image_id** – string, [<remote>:]<image>, the image to launch
- **instance_type** – string, type to use

- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – boolean, wait for instance to start
- **name** – string, what to call the instance
- **ephemeral** – boolean, ephemeral, otherwise persistent
- **network** – string, network name to use
- **storage** – string, storage name to use
- **profile_list** – list, profile(s) to use
- **config_dict** – dict, configuration values to pass
- **execute_via_ssh** – bool, optional, execute commands on the instance via SSH if True (the default)

Returns The created LXD instance object

Raises: ValueError on missing image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *arch='amd64'*)
Find the LXD fingerprint of the latest released image.

Parameters

- **release** – string, Ubuntu release to look for
- **arch** – string, architecture to use

Returns string, LXD fingerprint of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Take a snapshot of the passed in instance for use as image.

Parameters

- **instance** (LXDInstance) – The instance to create an image from
- **clean** – Whether to call cloud-init clean before creation
- **wait** – Whether to wait until before image is created before returning
- **name** – Name of the new image
- **stateful** – Whether to use an LXD stateful snapshot

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

pycloudlib.lxd.defaults module

LXD default values to be used by cloud and instance modules.

pycloudlib.lxd.instance module

LXD instance.

class pycloudlib.lxd.instance.**LXDInstance**(*name*, *key_pair=None*, *execute_via_ssh=True*, *series=None*, *ephemeral=None*)

Bases: *pycloudlib.instance.BaseInstance*

LXD backed instance.

__init__(*name*, *key_pair=None*, *execute_via_ssh=True*, *series=None*, *ephemeral=None*)

Set up instance.

Parameters

- **name** – name of instance
- **key_pair** – SSH key object
- **execute_via_ssh** – Boolean True to use ssh instead of lxc exec for all operations.
- **series** – Ubuntu release name: xenial, bionic etc.
- **ephemeral** – Boolean True if instance is ephemeral. If left unspecified, ephemeral type will be determined and cached by the ephemeral method.

add_network_interface() → str

Add nic to running instance.

clean()

Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()

Return console log.

Uses the '–show-log' option of console to get the console log from an instance.

Returns bytes of this instance's console

delete(*wait=True*)

Delete the current instance.

By default this will use the '–force' option to prevent the need to always stop the instance first. This makes it easier to work with ephemeral instances as well, which are deleted on stop.

Parameters **wait** – wait for delete

delete_snapshot(*snapshot_name*)

Delete a snapshot of the instance.

Parameters **snapshot_name** – the name to delete

edit(*key*, *value*)

Edit the config of the instance.

Parameters

- **key** – The config key to edit

- **value** – The new value to set the key to

ephemeral
Return boolean if ephemeral or not.

Will return False if unknown.

Returns boolean if ephemeral

execute(*command*, *stdin=None*, *description=None*, ***, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

Returns IP address assigned to instance.

Raises: PycloudlibTimeoutError when exhausting retries trying to parse lxc list for ip addresses.

is_vm
Return boolean if vm type or not.

Will return False if unknown.

Returns boolean if virtual-machine

local_snapshot(*snapshot_name*, *stateful=False*)
Create an LXD snapshot (not a launchable image).

Parameters

- **snapshot_name** – name to call snapshot
- **stateful** – boolean, stateful snapshot or not

name
Return instance name.

parse_ip (*query: dict*)
Return ip address from lxd query.

Returns None if no address found

pull_file (*remote_path*, *local_path*)
Pull file from an instance.

The remote path must be absolute path with LXD due to the way files are pulled off. Specifically, the format is 'name/path' with path assumed to start from '/'.

Parameters

- **remote_path** – path to remote file to pull down
- **local_path** – local path to put the file

push_file (*local_path*, *remote_path*)
Push file to an instance.

The remote path must be absolute path with LXD due to the way files are pulled off. Specifically, the format is 'name/path' with path assumed to start from '/'.

Parameters

- **local_path** – local path to file to push up
- **remote_path** – path to push file

remove_network_interface (*ip_address: str*)
Remove nic from running instance.

restart (*wait=True*, ***kwargs*)
Restart an instance.

restore (*snapshot_name*)
Restore instance from a specific snapshot.

Parameters **snapshot_name** – Name of snapshot to restore from

run_script (*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown (*wait=True*, *force=False*, ***kwargs*)
Shutdown instance.

Parameters

- **wait** – boolean, wait for instance to shutdown
- **force** – boolean, force instance to shutdown

snapshot (*snapshot_name*)
Create an image snapshot.

Snapshot is a bit of a misnomer here. Since "snapshot" in the context of clouds means "create a launchable container from this instance", we actually need to do a publish here. If you need the lxd "snapshot" functionality, use local_snapshot

Parameters **snapshot_name** – name to call snapshot

start(*wait=True*)
Start instance.

Parameters **wait** – boolean, wait for instance to fully start

state
Return current status of instance.

If unable to get status will return 'Unknown'.

Returns Reported status from lxc info

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete()
Wait for delete.

Not used for LXD.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_state(*desired_state: str*, *num_retries: int = 100*)
Wait for instance to reach desired state value.

Parameters

- **desired_state** – String representing one of lxc instance states seen by *lxc ls -s*. For example, ACTIVE, FROZEN, RUNNING, STOPPED
- **retries** – Integer for number of retry attempts before raising a PycloudlibTimeoutError.

wait_for_stop()
Wait for cloud instance to transition to stop state.

class pycloudlib.lxd.instance.**LXDVirtualMachineInstance**(*name*, *key_pair=None*, *execute_via_ssh=True*, *series=None*, *ephemeral=None*)

Bases: *pycloudlib.lxd.instance.LXDInstance*

LXD Virtual Machine backed instance.

__init__(*name*, *key_pair=None*, *execute_via_ssh=True*, *series=None*, *ephemeral=None*)
Set up instance.

Parameters

- **name** – name of instance
- **key_pair** – SSH key object
- **execute_via_ssh** – Boolean True to use ssh instead of lxc exec for all operations.

- **series** – Ubuntu release name: xenial, bionic etc.
- **ephemeral** – Boolean True if instance is ephemeral. If left unspecified, ephemeral type will be determined and cached by the ephemeral method.

add_network_interface() → str
Add nic to running instance.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Return console log.

Uses the '–show-log' option of console to get the console log from an instance.

Returns bytes of this instance's console

delete(*wait=True*)
Delete the current instance.

By default this will use the '–force' option to prevent the need to always stop the instance first. This makes it easier to work with ephemeral instances as well, which are deleted on stop.

Parameters **wait** – wait for delete

delete_snapshot(*snapshot_name*)
Delete a snapshot of the instance.

Parameters **snapshot_name** – the name to delete

edit(*key*, *value*)
Edit the config of the instance.

Parameters

- **key** – The config key to edit
- **value** – The new value to set the key to

ephemeral
Return boolean if ephemeral or not.

Will return False if unknown.

Returns boolean if ephemeral

execute(*command*, *stdin=None*, *description=None*, *, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

Returns IP address assigned to instance.

Raises: PycloudlibTimeoutError when exhausting retries trying to parse lxc list for ip addresses.

is_vm
Return boolean if vm type or not.

Will return False if unknown.

Returns boolean if virtual-machine

local_snapshot(*snapshot_name*, *stateful=False*)
Create an LXD snapshot (not a launchable image).

Parameters

- **snapshot_name** – name to call snapshot
- **stateful** – boolean, stateful snapshot or not

name
Return instance name.

parse_ip(*query: dict*)
Return ip address from lxd query.

Returns None if no address found

pull_file(*remote_path*, *local_path*)
Pull file from an instance.

The remote path must be absolute path with LXD due to the way files are pulled off. Specifically, the format is 'name/path' with path assumed to start from '/'.

Parameters

- **remote_path** – path to remote file to pull down
- **local_path** – local path to put the file

push_file(*local_path*, *remote_path*)
Push file to an instance.

The remote path must be absolute path with LXD due to the way files are pulled off. Specifically, the format is 'name/path' with path assumed to start from '/'.

Parameters

- **local_path** – local path to file to push up
- **remote_path** – path to push file

remove_network_interface(*ip_address: str*)
Remove nic from running instance.

restart(*wait=True*, ***kwargs*)
Restart an instance.

restore(*snapshot_name*)
Restore instance from a specific snapshot.

Parameters **snapshot_name** – Name of snapshot to restore from

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown(*wait=True*, *force=False*, ***kwargs*)
Shutdown instance.

Parameters

- **wait** – boolean, wait for instance to shutdown
- **force** – boolean, force instance to shutdown

snapshot(*snapshot_name*)
Create an image snapshot.

Snapshot is a bit of a misnomer here. Since "snapshot" in the context of clouds means "create a launchable container from this instance", we actually need to do a publish here. If you need the lxd "snapshot" functionality, use local_snapshot

Parameters **snapshot_name** – name to call snapshot

start(*wait=True*)
Start instance.

Parameters **wait** – boolean, wait for instance to fully start

state
Return current status of instance.

If unable to get status will return 'Unknown'.

Returns Reported status from lxc info

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete()
Wait for delete.

Not used for LXD.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_state(*desired_state: str*, *num_retries: int = 100*)
Wait for instance to reach desired state value.

Parameters

- **desired_state** – String representing one of lxc instance states seen by *lxc ls -s*. For example, ACTIVE, FROZEN, RUNNING, STOPPED
- **retries** – Integer for number of retry attempts before raising a PycloudlibTimeoutError.

wait_for_stop()
Wait for cloud instance to transition to stop state.

pycloudlib.oci package

Main OCI module __init__.

Submodules

pycloudlib.oci.cloud module

OCI Cloud type.

class pycloudlib.oci.cloud.**OCI**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *availability_domain: Optional[str] = None*, *compartment_id: Optional[str] = None*, *config_path: Optional[str] = None*, *config_dict: Optional[str] = None*)
Bases: *pycloudlib.cloud.BaseCloud*

OCI (Oracle) cloud class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *availability_domain: Optional[str] = None*, *compartment_id: Optional[str] = None*, *config_path: Optional[str] = None*, *config_dict: Optional[str] = None*)
Initialize the connection to OCI.

OCI must be initialized on the CLI first: https://github.com/cloud-init/qa-scripts/blob/master/doc/launching-oracle.md

Parameters

- **tag** – Name of instance
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **compartment_id** – A compartment found at https://console.us-phoenix-1.oraclecloud.com/a/identity/compartments
- **availability_domain** – One of the availability domains from: 'oci iam availability-domain list'
- **config_path** – Path of OCI config file

- **config_dict** – A dictionary containing the OCI config. Overrides the values from config_path

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, *operating_system: str = 'Canonical Ubuntu'*, ***kwargs*)
Get the daily image.

OCI just has periodic builds, so "released" and "daily" don't really make sense here. Just call the same code for both.

Should be equivalent to the cli call: oci compute image list –operating-system="Canonical Ubuntu" –operating-system-version="<xx.xx>" –sort-by='TIMECREATED' –sort-order='DESC'

Parameters

- **release** – string, Ubuntu release to look for
- **operating_system** – string, Operating system to use
- ****kwargs** – dictionary of other arguments to pass to list_images

Returns string, id of latest image

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters **image_id** – string, id of the image to delete

get_instance(*instance_id*, ***kwargs*) → pycloudlib.oci.instance.OciInstance
Get an instance by id.

Parameters

- **instance_id** – ocid of the instance
- ****kwargs** – dictionary of other arguments to pass to get_instance

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='VM.Standard2.1'*, *user_data=None*, *wait=True*, ***, *retry_strategy=None*, ***kwargs*) → pycloudlib.oci.instance.OciInstance
Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type of instance to create. https://docs.cloud.oracle.com/en-us/iaas/Content/Compute/References/computeshapes.htm
- **user_data** – used by Cloud-Init to run custom scripts or provide custom Cloud-Init configuration
- **wait** – wait for instance to be live
- **retry_strategy** – a retry strategy from oci.retry module to apply for this operation

- ****kwargs** – dictionary of other arguments to pass as LaunchInstanceDetails

Returns An instance object to use to manipulate the instance further.

Raises: ValueError on invalid image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, *operating_system='Canonical Ubuntu'*)
Get the released image.

OCI just has periodic builds, so "released" and "daily" don't really make sense here. Just call the same code for both

Parameters

- **release** – string, Ubuntu release to look for
- **operating_system** – string, operating system to use

Returns string, id of latest image

snapshot(*instance*, *clean=True*, *name=None*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot
- **name** – (Optional) Name of created image

Returns An image object

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

pycloudlib.oci.instance module

OCI instance.

class pycloudlib.oci.instance.**OciInstance**(*key_pair*, *instance_id*, *compartment_id*, *availability_domain*, *oci_config=None*)
Bases: *pycloudlib.instance.BaseInstance*

OCI backed instance.

__init__(*key_pair*, *instance_id*, *compartment_id*, *availability_domain*, *oci_config=None*)
Set up the instance.

Parameters

- **key_pair** – A KeyPair for SSH interactions
- **instance_id** – The instance id representing the cloud instance

- **compartment_id** – A compartment found at https://console.us-phoenix-1.oraclecloud.com/a/identity/compartments
- **availability_domain** – One of the availability domains from: 'oci iam availability-domain list'
- **oci_config** – OCI configuration dictionary

add_network_interface() → str
Add network interface to running instance.

Creates a nic and attaches it to the instance. This is effectively a hot-add of a network device. Returns the IP address of the added network interface as a string.

Note: It assumes the associated compartment has at least one subnet and creates the vnic in the first encountered subnet.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Not currently implemented.

delete(*wait=True*)
Delete the instance.

Parameters **wait** – wait for instance to be deleted

execute(*command*, *stdin=None*, *description=None*, *, *use_sudo=False*, *no_log=False*, **kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

instance_data
Return JSON formatted details from OCI about this instance.

ip
Return IP address of instance.

name
Return the instance name.

pull_file(*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file(*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface(*ip_address: str*)
Remove network interface based on IP address.

Find the NIC from the IP, detach from the instance.

Note: In OCI, detaching triggers deletion.

restart(*wait=True*, ***kwargs*)
Restart an instance.

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown(*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance to shutdown

start(*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete()
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

pycloudlib.oci.utils module

Utilities for OCI images and instances.

pycloudlib.oci.utils.**get_subnet_id**(*network_client: oci.core.VirtualNetworkClient*, *compartment_id: str*, *availability_domain: str*) → str
Get a subnet id linked to *availability_domain*.

From specified compartment select the first subnet linked to *availability_domain* or the first one.

Parameters

- **network_client** – Instance of VirtualNetworkClient.
- **compartment_id** – Compartment where the subnet has to belong
- **availability_domain** – Domain to look for subnet id in.

Returns The updated version of the current_data

Raises

- *Exception* if unable to determine *subnet_id* for
- *availability_domain*

pycloudlib.oci.utils.**wait_till_ready**(*func*, *current_data*, *desired_state*, *sleep_seconds=1000*)
Wait until the results of function call reach a desired lifecycle state.

Parameters

- **func** – The function to call
- **current_data** – Structure containing the initial id and lifecycle state
- **desired_state** – Desired value of "lifecycle_state"
- **sleep_seconds** – How long to wait in seconds

Returns The updated version of the current_data

pycloudlib.openstack package

OpenStack handling code for pycloudlib.

Submodules

pycloudlib.openstack.cloud module

Openstack cloud type.

class pycloudlib.openstack.cloud.**Openstack**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *network: Optional[str] = None*)

Bases: *pycloudlib.cloud.BaseCloud*

Openstack cloud class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, ***, *network: Optional[str] = None*)

Initialize the connection to openstack.

Requires valid pre-configured environment variables or clouds.yaml. See https://docs.openstack.org/python-openstackclient/pike/configuration/index.html

Parameters

- **tag** – Name of instance
- **timestamp_suffix** – bool set True to append a timestamp suffix to the tag
- **config_file** – path to pycloudlib configuration file
- **network** – Name of the network to use (from openstack network list)

create_key_pair()

Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***kwargs*)

Not supported for openstack.

delete_image(*image_id*, ***kwargs*)

Delete an image.

Parameters **image_id** – string, id of the image to delete

get_instance(*instance_id*, ***kwargs*) → pycloudlib.openstack.instance.OpenstackInstance

Get an instance by id.

Parameters **instance_id** – ID of instance to get

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)

Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id*, *instance_type='m1.small'*, *user_data=''*, *wait=True*, ***kwargs*) → pycloudlib.openstack.instance.OpenstackInstance

Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type (flavor) of instance to create
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – wait for instance to be live
- ****kwargs** – dictionary of other arguments to pass to launch

Returns An instance object to use to manipulate the instance further.

Raises: ValueError on invalid image_id

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***kwargs*)
Not supported for openstack.

snapshot(*instance*, *clean=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

pycloudlib.openstack.errors module

Module containing errors specific to openstack.

exception pycloudlib.openstack.errors.**OpenStackError**
Bases: *pycloudlib.errors.PycloudlibException*

OpenStack exception root.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

exception pycloudlib.openstack.errors.**OpenStackFlavorNotFound**
Bases: *pycloudlib.openstack.errors.OpenStackError*

Raised when an OpenStack's flavor is not found.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

pycloudlib.openstack.instance module

Openstack instance type.

class pycloudlib.openstack.instance.**OpenstackInstance**(*key_pair*, *instance_id*, *network_id*, *connection=None*)

Bases: *pycloudlib.instance.BaseInstance*

Openstack instance object.

__init__(*key_pair*, *instance_id*, *network_id*, *connection=None*)

Set up the instance.

Parameters

- **key_pair** – A KeyPair for SSH interactions
- **instance_id** – The instance id representing the cloud instance
- **network_id** – if of the network this instance was created on
- **connection** – The connection used to create this instance. If None, connection will be created.

add_network_interface() → str

Add nic to running instance.

Returns IP address in string form

clean()

Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()

Return the instance console log.

delete(*wait=True*)

Delete the instance.

Parameters **wait** – wait for instance to be deleted

execute(*command*, *stdin=None*, *description=None*, *, *use_sudo=False*, *no_log=False*, **kwargs*)

Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()

Get the instance boot_id.

Returns string with the boot UUID

install (*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return instance name.

pull_file (*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file (*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface (*ip_address: str*)
Remove nic from running instance.

restart (*wait=True*, ***kwargs*)
Restart an instance.

run_script (*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown (*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance to shutdown

start (*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update ()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete()
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

Submodules

pycloudlib.cloud module

Base class for all other clouds to provide consistent set of functions.

class pycloudlib.cloud.**BaseCloud**(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, *required_values: Optional[Sequence[Optional[Any]]] = None*)
Bases: abc.ABC

Base Cloud Class.

__init__(*tag: str*, *timestamp_suffix: bool = True*, *config_file: Union[pathlib.Path, _io.StringIO, None] = None*, *required_values: Optional[Sequence[Optional[Any]]] = None*)
Initialize base cloud class.

Parameters

- **tag** – string used to name and tag resources with
- **timestamp_suffix** – Append a timestamped suffix to the tag string.
- **config_file** – path to pycloudlib configuration file

create_key_pair()
Create and set a ssh key pair for a cloud instance.

Returns A tuple containing the public and private key created

daily_image(*release: str*, ***kwargs*)
ID of the latest daily image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest daily image ID for the specified release.

delete_image(*image_id*, ***kwargs*)
Delete an image.

Parameters

- **image_id** – string, id of the image to delete
- ****kwargs** – dictionary of other arguments to pass to delete_image

get_instance(*instance_id*, ***kwargs*) → pycloudlib.instance.BaseInstance
Get an instance by id.

Parameters **instance_id** –

Returns An instance object to use to manipulate the instance further.

image_serial(*image_id*)
Find the image serial of the latest daily image for a particular release.

Parameters **image_id** – string, Ubuntu image id

Returns string, serial of latest image

launch(*image_id: str*, *instance_type=None*, *user_data=None*, *wait: bool = True*, ***kwargs*) → pycloudlib.instance.BaseInstance
Launch an instance.

Parameters

- **image_id** – string, image ID to use for the instance
- **instance_type** – string, type of instance to create
- **user_data** – used by cloud-init to run custom scripts/configuration
- **wait** – wait for instance to be live
- ****kwargs** – dictionary of other arguments to pass to launch

Returns An instance object to use to manipulate the instance further.

list_keys()
List ssh key names present on the cloud for accessing instances.

Returns A list of strings of key pair names accessible to the cloud.

released_image(*release*, ***kwargs*)
ID of the latest released image for a particular release.

Parameters **release** – The release to look for

Returns A single string with the latest released image ID for the specified release.

snapshot(*instance*, *clean=True*, ***kwargs*)
Snapshot an instance and generate an image from it.

Parameters

- **instance** – Instance to snapshot
- **clean** – run instance clean method before taking snapshot

Returns An image id

use_key(*public_key_path*, *private_key_path=None*, *name=None*)
Use an existing key.

Parameters

- **public_key_path** – path to the public key to upload
- **private_key_path** – path to the private key
- **name** – name to reference key by

class pycloudlib.cloud.**ImageType**
Bases: enum.Enum

Allowed image types when launching cloud images.

GENERIC = 'generic'

```
PRO = 'Pro'

PRO_FIPS = 'Pro FIPS'
```

pycloudlib.config module

Deal with configuration file.

class pycloudlib.config.**Config**
Bases: dict

Override dict to allow raising a more meaningful KeyError.

__init__
Initialize self. See help(type(self)) for accurate signature.

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys()
Create a new dictionary with keys from iterable and values set to value.

get()
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault()
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

pycloudlib.config.**parse_config**(*config_file: Union[pathlib.Path, _io.StringIO, None] = None*)
→ MutableMapping[str, Any]
Find the relevant TOML, load, and return it.

pycloudlib.constants module

Constants.

pycloudlib.errors module

Module containing pycloudlib errors.

Each cloud can have specific errors, please refer to each *pycloudlib.<cloud>.errors* module.

exception pycloudlib.errors.**CloudError**
Bases: *pycloudlib.errors.PycloudlibException*

Represents errors coming from Cloud's SDKs.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

exception pycloudlib.errors.**CloudSetupError**
Bases: *pycloudlib.errors.PycloudlibException*

Raised if there is some problem with a pycloudlib's Cloud set up.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

exception pycloudlib.errors.**ImageNotFoundError**(**args*, ***kwargs*)
Bases: *pycloudlib.errors.ResourceNotFoundError*

Sepecialized's *ResourceNotFoundError* for images.

__init__(**args*, ***kwargs*)
Init method.

Parameters

- **resource_type** – Instance of *ResourceType*
- **resource_id** – Resource's id
- **resource_type** – Resource's name

args

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

exception pycloudlib.errors.**InstanceNotFoundError**(**args*, ***kwargs*)
Bases: *pycloudlib.errors.ResourceNotFoundError*

Sepecialized's *ResourceNotFoundError* for instances.

__init__(**args*, ***kwargs*)
Init method.

Parameters

- **resource_type** – Instance of *ResourceType*
- **resource_id** – Resource's id
- **resource_type** – Resource's name

args

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

exception pycloudlib.errors.**NetworkNotFoundError**(**args*, ***kwargs*)
Bases: *pycloudlib.errors.ResourceNotFoundError*

Sepecialized's *ResourceNotFoundError* for networks.

__init__(**args*, ***kwargs*)
Init method.

Parameters

- **resource_type** – Instance of *ResourceType*
- **resource_id** – Resource's id
- **resource_type** – Resource's name

args

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

exception pycloudlib.errors.**PycloudlibError**
Bases: *pycloudlib.errors.PycloudlibException*

Error that doesn't fall in any of the other categories.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

exception pycloudlib.errors.**PycloudlibException**
Bases: Exception

Root pycloudlib exception.

This exception is not meant to be raised by pycloudlib. The intention is that every custom pycloudlib exception will inherit from this one, allowing client code to catch any exception by catching this one.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

exception pycloudlib.errors.**PycloudlibTimeoutError**
Bases: *pycloudlib.errors.PycloudlibException*

Timeout error.

__init__
Initialize self. See help(type(self)) for accurate signature.

args

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

exception pycloudlib.errors.**ResourceNotFoundError**(*resource_type: pycloudlib.errors.ResourceType*, *resource_id: Optional[str] = None*, *resource_name: Optional[str] = None*, ***kwargs*)

Bases: *pycloudlib.errors.PycloudlibException*

Raised when a resource is not found.

```
>>> e = ResourceNotFoundError(ResourceType.IMAGE, "id-123")
>>> e.resource_id
'id-123'
>>> raise e  # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
pycloudlib.errors.ResourceNotFoundError: Could not locate the resource type␣
↪`image`: id=id-123
```

__init__(*resource_type: pycloudlib.errors.ResourceType*, *resource_id: Optional[str] = None*, *resource_name: Optional[str] = None*, ***kwargs*)

Init method.

Parameters

- **resource_type** – Instance of *ResourceType*
- **resource_id** – Resource's id
- **resource_type** – Resource's name

args

with_traceback()

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

class pycloudlib.errors.**ResourceType**

Bases: enum.Enum

Represent types of resources.

IMAGE = 1

INSTANCE = 2

NETWORK = 3

pycloudlib.instance module

Base class for all instances to provide consistent set of functions.

class pycloudlib.instance.**BaseInstance**(*key_pair*)

Bases: abc.ABC

Base instance object.

__init__(*key_pair*)

Set up instance.

add_network_interface() → str

Add nic to running instance.

clean()
Clean an instance to make it look prestine.

This will clean out specifically the cloud-init files and system logs.

console_log()
Return the instance console log.

Raises NotImplementedError if the cloud does not support fetching the console log for this instance.

delete(*wait=True*)
Delete the instance.

Parameters **wait** – wait for instance to be deleted

execute(*command*, *stdin=None*, *description=None*, ***, *use_sudo=False*, *no_log=False*, ***kwargs*)
Execute command in instance, recording output, error and exit code.

Assumes functional networking and execution with the target filesystem being available at /.

Parameters

- **command** – the command to execute as root inside the image. If command is a string, then it will be executed as: *['sh', '-c', command]*
- **stdin** – bytes content for standard in
- **description** – purpose of command
- **use_sudo** – boolean to run the command as sudo

Returns Result object

Raises SSHException if there are any problem with the ssh connection

get_boot_id()
Get the instance boot_id.

Returns string with the boot UUID

install(*packages*)
Install specific packages.

Parameters **packages** – string or list of package(s) to install

Returns result from install

ip
Return IP address of instance.

name
Return instance name.

pull_file(*remote_path*, *local_path*)
Copy file at 'remote_path', from instance to 'local_path'.

Parameters

- **remote_path** – path on remote instance
- **local_path** – local path

Raises SSHException if there are any problem with the ssh connection

push_file(*local_path*, *remote_path*)
Copy file at 'local_path' to instance at 'remote_path'.

Parameters

- **local_path** – local path
- **remote_path** – path on remote instance

Raises SSHException if there are any problem with the ssh connection

remove_network_interface(*ip_address: str*)
Remove nic from running instance.

restart(*wait=True*, ***kwargs*)
Restart an instance.

run_script(*script*, *description=None*)
Run script in target and return stdout.

Parameters

- **script** – script contents
- **description** – purpose of script

Returns result from script execution

Raises SSHException if there are any problem with the ssh connection

shutdown(*wait=True*, ***kwargs*)
Shutdown the instance.

Parameters **wait** – wait for the instance to shutdown

start(*wait=True*)
Start the instance.

Parameters **wait** – wait for the instance to start.

update()
Run apt-get update/upgrade on instance.

Returns result from upgrade

wait()
Wait for instance to be up and cloud-init to be complete.

wait_for_delete(***kwargs*)
Wait for instance to be deleted.

wait_for_restart(*old_boot_id*)
Wait for instance to be restarted and cloud-init to be complete.

old_boot_id is the boot id prior to restart

wait_for_stop()
Wait for instance stop.

pycloudlib.key module

Base Key Class.

class pycloudlib.key.**KeyPair**(*public_key_path*, *private_key_path=None*, *name=None*)
Bases: object

Key Class.

__init__ (*public_key_path*, *private_key_path=None*, *name=None*)
Initialize key class to generate key or reuse existing key.

The public key path is given then the key is stored and the private key is assumed to be named the same minus '.pub' otherwise the user should also specify the private key path.

Parameters

- **public_key_path** – Path to public key
- **private_key_path** – Path to private key
- **name** – Name to reference key by in clouds

public_key_content
Read the contents of the public key.

Returns output of public key

pycloudlib.result module

Base Result Class.

class pycloudlib.result.**Result**(*stdout*, *stderr=''*, *return_code=0*)
Bases: str

Result Class.

__init__ (*stdout*, *stderr=''*, *return_code=0*)
Initialize class.

capitalize()
Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()
Return a version of the string suitable for caseless comparisons.

center()
Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count (*sub*[, *start*[, *end*]]) → int
Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode()
Encode the string using the codec registered for encoding.

encoding The encoding in which to encode the string.

errors The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith (*suffix*[, *start*[, *end*]]) → bool
Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs()
Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

failed
Return boolean if result was failure.

find(*sub*[, *start*[, *end*]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str
Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()
Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()
Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()
Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()
Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()
Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()
Return True if the string is a valid Python identifier, False otherwise.

Use keyword.iskeyword() to test for reserved identifiers such as "def" and "class".

islower()
Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()
Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()
Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()
Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()
Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()
Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join()
Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

ljust()
Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()
Return a copy of the string converted to lowercase.

lstrip()
Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans()
Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

ok
Return boolean if result was success.

partition()
Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

replace()
Return a copy with all occurrences of substring old replaced by new.

count Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust()
Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition()
Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit()
Return a list of the words in the string, using sep as the delimiter string.

sep The delimiter according which to split the string. None (the default value) means split according to any whitespace, and discard empty strings from the result.

maxsplit Maximum number of splits to do. -1 (the default value) means no limit.

Splits are done starting at the end of the string and working to the front.

rstrip()
Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split()
Return a list of the words in the string, using sep as the delimiter string.

sep The delimiter according which to split the string. None (the default value) means split according to any whitespace, and discard empty strings from the result.

maxsplit Maximum number of splits to do. -1 (the default value) means no limit.

splitlines()
Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip()
Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()
Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()
Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate()
Replace each character in the string using the given translation table.

table Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

upper()
Return a copy of the string converted to uppercase.

zfill()
Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

pycloudlib.util module

Helpers for shell string and processing.

pycloudlib.util.**chmod**(*path*, *mode*)
Run chmod on a file or directory.

Parameters

- **path** – string of path to run on
- **mode** – int of mode to apply

pycloudlib.util.**get_query_param**(*uri: str*, *param: str*) → list
Extract query query params of *param* from *uri*.

```
>>> url = "https://cloud.com/v1/vpcs?limit=1&start=r134-fe06d70f"
>>> get_query_param(url, "start")
['r134-fe06d70f']
```

pycloudlib.util.**get_query_params**(*uri: str*) → Dict[str, list]
Extract query params from *uri*.

```
>>> url = "https://cloud.com/v1/vpcs?limit=1&start=r134-fe06d70f"
>>> get_query_params(url)
{'limit': ['1'], 'start': ['r134-fe06d70f']}
```

pycloudlib.util.**get_timestamped_tag**(*tag*)
Create tag with current timestamp.

Parameters **tag** – string, Base tag to be used

Returns An updated tag with current timestamp

pycloudlib.util.**is_writable_dir**(*path*)
Make sure dir is writable.

Parameters **path** – path to determine if writable

Returns boolean with result

pycloudlib.util.**mkdtemp**(*prefix='pycloudlib'*)
Make a temporary directory.

Parameters **prefix** – optional, temproary dir name prefix (default: pycloudlib)

Returns tempfile object that was created

pycloudlib.util.**rmfile**(*path*)
Delete a file.

Parameters **path** – run unlink on specific path

pycloudlib.util.**shell_pack**(*cmd*)
Return a string that can shuffled through 'sh' and execute cmd.

In Python subprocess terms: check_output(cmd) == check_output(shell_pack(cmd), shell=True)

Parameters **cmd** – list or string of command to pack up

Returns base64 encoded string

pycloudlib.util.**shell_quote**(*cmd*)
Quote a shell string.

Parameters **cmd** – command to quote

Returns quoted string

pycloudlib.util.**shell_safe**(*cmd*)
Produce string safe shell string.

Create a string that can be passed to $(set – <string>) to produce the same array that cmd represents.

Internally we utilize 'getopt's ability/knowledge on how to quote strings to be safe for shell. This implementation could be changed to be pure python. It is just a matter of correctly escaping or quoting characters like: ' " ^ & $; ( ) ...

Parameters **cmd** – command as a list

Returns shell safe string

pycloudlib.util.**subp**(*args*, *data=None*, *env=None*, *shell=False*, *rcs=(0,)*, *shortcircuit_stdin=True*)
Subprocess wrapper.

Parameters

- **args** – command to run
- **data** – data to pass
- **env** – optional env to use
- **shell** – optional shell to use
- **rcs** – tuple of successful exit codes, default: (0)
- **shortcircuit_stdin** – bind stdin to /dev/null if no data is given

Returns Tuple of out, err, return_code

pycloudlib.util.**touch**(*path*, *mode=None*)

Ensure a directory exists with a specific mode, it not create it.

Parameters

- **path** – path to directory to create
- **mode** – optional, mode to set directory to

pycloudlib.util.**update_nested**(*mapping*, *update*)

Update mapping with update value at given update key.

Example

original_dict = {'a': {'b': {'c': 'd'}}} update = {'a': {'b': {'c': 'e'}}} update_nested(original_dict, update) original_dict == {'a': {'b': {'c': 'e'}}}

pycloudlib.util.**validate_tag**(*tag*)

Ensure tag will work as name for clouds that use it.

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z