pycloudlib Documentation

Joshua Powers

Sep 13, 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.

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:latest"
```

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

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:latest',
    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:latest', 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:latest')
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')
```

(continues on next page)

(continued from previous page)

```
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 VMWare

The VMWare support in pycloudlib is specific to vSphere. In particular, vSphere 7 was tested.

5.8.1 Prerequisites

VMWare usage in Pycloudlib requires the govc command line tool to be available on the PATH. See VMWare docs for installation information.

5.8.2 Available Images

To create new instances, pycloudlib will clone an existing VM within vSphere that is designated as the image source. In order to qualify, the VM must meet the following requirements:

- A standard (non-template) VM.
- Powered off
- In the same folder that new VMs will be deployed to (see `folder` in `pycloudlib.toml`)
- Have the "InjectOvfEnv" setting be `false`.
- Be named appropriately: `TEMPLATE-cloud-init-<release>`

As of this writing, `TEMPLATE-cloud-init-focal` and `TEMPLATE-cloud-init-jammy` are valid source VMs.

To create the Ubuntu-based source images, the following procedure was followed for a Jammy image:

- Download the `.ova` for the release from the release server
- `govc import.spec ubuntu-jammy-server-cloudimg-amd64.ova | python -m json.tool > ubuntu.json`
- Modify the json file appropriately
- `govc import.ova -options=ubuntu.json ./ubuntu-jammy-server-cloudimg-amd64.ova`

Example ubuntu.json:

```
{
    "DiskProvisioning": "thin",
    "IPAllocationPolicy": "dhcpPolicy",
    "IPProtocol": "IPv4",
    "PropertyMapping": [
```

(continues on next page)

(continued from previous page)

```
        {
            "Key": "instance-id",
            "Value": ""
        },
        {
            "Key": "hostname",
            "Value": ""
        },
        {
            "Key": "seedfrom",
            "Value": ""
        },
        {
            "Key": "public-keys",
            "Value": ""
        },
        {
            "Key": "user-data",
            "Value": ""
        },
        {
            "Key": "password",
            "Value": ""
        }
    ],
    "NetworkMapping": [
        {
            "Name": "VM Network",
            "Network": "VLAN_2763"
        }
    ],
    "MarkAsTemplate": false,
    "PowerOn": false,
    "InjectOvfEnv": false,
    "WaitForIP": false,
    "Name": "TEMPLATE-cloud-init-jammy"
}
```

5.8.3 SSH Keys

To avoid cloud-init detecting an instance as an OVF datasource, passing a public key through ovf xml is not supported. Rather, when the instance is created, the pycloudlib managed ssh public key is added to the cloud-config user data of the instance. This means that the user data on the launched instance will always contain an extra public key compared to what was passed to pycloudlib.

5.8.4 Blocking calls

Since calls to `govc` are blocking, specifying `wait=False` to enable non-blocking calls will not work.

5.9 EC2

```python
#!/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.
    """
    with ec2.launch(daily, instance_type="m4.xlarge") as instance:
        instance.wait()
        instance.add_network_interface()
        instance.add_network_interface()

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


def launch_multiple(ec2, daily):
    """Launch multiple instances.

    How to quickly launch multiple instances with EC2. This prevents
    waiting for the instance to start each time.
    """
    instances = []
    for _ in range(3):
        instances.append(ec2.launch(daily))

    for instance in instances:
        instance.wait()

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

    for instance in instances:
        instance.wait_for_delete()


def snapshot(ec2, daily):
    """Create a snapshot from a customized image and launch it."""
    with ec2.launch(daily) as instance:
        instance.wait()
        instance.execute("touch custom_config_file")

        image = ec2.snapshot(instance)
        new_instance = ec2.launch(image)
        new_instance.wait()
```

(continues on next page)

(continued from previous page)

```
        new_instance.execute("ls")

        new_instance.delete()
        ec2.delete_image(image)


def custom_vpc(ec2, daily):
    """Launch instances using a custom VPC."""
    vpc = ec2.get_or_create_vpc(name="test-vpc")
    with ec2.launch(daily, vpc=vpc) as instance:
        instance.wait()
        instance.execute("whoami")

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


def launch_basic(ec2, daily):
    """Show basic functionality on instances.

    Simple launching of an instance, run a command, and delete.
    """
    with ec2.launch(daily) as instance:
        instance.wait()
        instance.console_log()
        print(instance.execute("lsb_release -a"))

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

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


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


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


def handle_ssh_key(ec2, key_name):
```

(continues on next page)

(continued from previous page)

```
    """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.
    """
    with pycloudlib.EC2(tag="examples") as ec2:
        key_name = "test-ec2"
        handle_ssh_key(ec2, key_name)

        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()
```

5.10 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."""

```

(continues on next page)

(continued from previous page)

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


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


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


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

(continues on next page)

(continued from previous page)

```
        manage_ssh_key(gce)

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


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

5.11 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."""
    with ibm.launch(daily) as instance:
        instance.wait()
        instance.execute("touch custom_config_file")

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

    ibm.delete_image(image)


def custom_vpc(ibm, daily):
    """Launch instances using a custom VPC."""
    vpc = ibm.get_or_create_vpc(name="test-vpc")
    with ibm.launch(daily, vpc=vpc) as instance:
        instance.wait()
        instance.execute("whoami")

    # 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.
    """
    with ibm.launch(daily, instance_type=instance_type) as instance:
        instance.wait()
        print(instance.execute("lsb_release -a"))
```

(continues on next page)

(continued from previous page)

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

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


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(
        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.
    """
    with pycloudlib.IBM(tag="examples") as ibm:
        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()
```

5.12 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.
    """
    with pycloudlib.LXDContainer("example-snapshot") as lxd:
        with lxd.launch(
            name="pycloudlib-snapshot-base", image_id=RELEASE
        ) as inst:
            inst.wait()
            snapshot_name = "snapshot"
            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
    """
    with pycloudlib.LXDContainer("example-image-snapshot") as lxd:
        with lxd.launch(
            name="pycloudlib-snapshot-base",
            image_id=RELEASE,
            ephemeral=ephemeral_instance,
        ) as inst:
            inst.wait()
            inst.execute("touch snapshot-test.txt")
            print("Base instance output: {}".format(inst.execute("ls")))
```

(continues on next page)

(continued from previous page)

```
            snapshot_image = lxd.snapshot(instance=inst)

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


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.
    """
    with pycloudlib.LXDContainer("example-modify") as lxd:
        with lxd.init("pycloudlib-modify-inst", RELEASE) as inst:
            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")
            inst.execute("cat /tmp/uptime_2")


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)
        instances.append(inst)

    for instance in instances:
        instance.wait()

    for instance in instances:
        instance.delete()


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

(continues on next page)

(continued from previous page)

```
    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",
        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",
        },
    )
```

(continues on next page)

(continued from previous page)

```
    lxd.delete_instance("pycloudlib-privileged")


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

        name = "pycloudlib-daily"
        with lxd.launch(name=name, image_id=RELEASE) as inst:
            inst.wait()
            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)

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


def launch_virtual_machine():
    """Demonstrate launching virtual machine scenario."""
    with pycloudlib.LXDVirtualMachine("example-vm") as lxd:
        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"
        with lxd.launch(name=name, image_id=image_id) as inst:
            inst.wait()
            print("Is vm: {}".format(inst.is_vm))
```

(continues on next page)

(continued from previous page)

```
            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()


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()
```

5.13 OCI

```
#!/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.
    """
```

(continues on next page)

(continued from previous page)

```
    with pycloudlib.OCI(
        "oracle-test",
        availability_domain=availability_domain,
        compartment_id=compartment_id,
    ) as client:
        with client.launch(
            image_id=client.released_image("focal"),
            user_data=b64encode(cloud_config.encode()).decode(),
        ) as instance:
            instance.wait()
            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)

        with client.launch(image_id=snapshotted_image_id) as new_instance:
            new_instance.wait()
            new_instance.execute("whoami")


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)
```

5.14 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

5.14.1 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.
```

(continues on next page)

(continued from previous page)

```
#
# 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
# 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
```

(continues on next page)

(continued from previous page)

```
[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]

[vmware]
# These are likely defined as environment variables if using govc. They correspond to:
# GOVC_URL
# GOVC_USERNAME
# GOVC_PASSWORD
# GOVC_DATACENTER
# GOVC_DATASTORE
# GOVC_FOLDER
# GOVC_INSECURE
#
# respectively.
server = ""
username = ""
password = ""
datacenter = ""
datastore = ""
folder = ""  # The folder to place new VMs as well as to find TEMPLATE VMs
insecure_transport = false
# 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
```

5.15 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.

5.15.1 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.

5.15.2 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.

5.15.3 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` |

5.15.4 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.

5.15.5 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'
```

5.16 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.

5.16.1 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.

5.17 Resource Cleanup

By default, pycloudlib will **not** automatically cleanup created resources because there are use cases for inspecting resources launched by pycloudlib after pycloudlib has exited.

5.17.1 Performing Cleanup

The easiest way to ensure cleanup happens is to use the `cloud` and `instance` context managers. For example, using EC2:

```
from pycloudlib.ec2.cloud import EC2


with EC2(tag="example") as cloud:
    with cloud.launch("your-ami") as instance:
        instance.wait()
        output = instance.execute("cat /etc/lsb-release").stdout


print(output)
```

When the context manager exits (even if due to an exception), all resources that were created during the lifetime of the `Cloud` or `Instance` object will automatically be cleaned up. Any exceptions raised during the cleanup process will be raised.

Alternatively, if you don't want to use context managers, you can manually cleanup all resources using the `.clean()` method on `Cloud` objects and the `.delete()` method on `Instance` objects. For example, using EC2:

```
from pycloudlib.ec2.cloud import EC2


cloud = EC2(tag="example")
instance = cloud.launch("your-ami")
instance.wait()
instance.execute("cat /etc/lsb-release").stdout

instance_cleanup_exceptions: List[Exception] = instance.delete()
cloud_cleanup_exceptions: List[Exception] = cloud.clean()
```

Things to note:

- Exceptions that occur during cleanup aren't automatically raised and are instead returned. This is to is to prevent a failure in one stage of cleanup from affecting another.
- Resources can still leak if an exception is raised between creating the object and cleaning it up. To ensure resources are not leaked, the body of code between launch and cleanup must be wrapped in an exception handler.

Because of these reasons, the context manager approach should be preferred.

5.18 Contributing

This document describes how to contribute changes to pycloudlib.

5.18.1 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
```

5.18.2 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`

5.18.3 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.

5.18.4 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
```

5.19 Maintainer Notes

5.19.1 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

5.20 Design

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

5.20.1 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.

5.20.2 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

5.20.3 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.

5.20.4 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.

5.20.5 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

5.21 API

5.21.1 pycloudlib

pycloudlib package

Subpackages

pycloudlib.azure package

Subpackages