python-cratonclient Documentation

Release

OpenStack Foundation

Apr 12, 2017

Contents

Contents:

CHAPTER 1

python-cratonclient

Craton API Client and Command-line Utility

Please fill here a long description which must be at least 3 lines wrapped on 80 cols, so that distribution package maintainers can use it in their packages. Note that this is a hard requirement.

- Free software: Apache license
- Documentation: https://python-cratonclient.readthedocs.io
- Source: http://git.openstack.org/cgit/openstack/python-cratonclient
- Bugs: http://bugs.launchpad.net/python-cratonclient

Features

- TODO

CHAPTER 2

Installation

Note: There has not been a stable release of python-cratonclient to PyPI yet. To install the current version in development use:

```
pip install git+https://git.openstack.org/openstack/python-cratonclient
```

At the command line:

```
$ pip install python-cratonclient
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv python-cratonclient
$ pip install python-cratonclient
```

CHAPTER 3

Contributing

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

> http://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

> http://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

> https://bugs.launchpad.net/python-cratonclient

CHAPTER 4

Craton CLI User Guide

After installing `python-cratonclient` and `craton` binary should be added to our PATH. To use the craton command-line client, we need the following information:

- URL to speak to Craton with
- Username to use to authenticate to Craton
- Password to use to authenticate to Craton
- Project ID to use to communicate with Craton

These items need to be provided to the craton command-line client. We can pass these as command-line arguments:

```
$ craton --craton-url <craton-url> \
         --os-username <username> \
         --os-password <password> \
         --os-project-id <project-id>
```

These parameters may also be provided via environment variables. We can create a file, similar to OpenStack's `openrc` file that contains:

```
# ~/cratonrc
export CRATON_URL="<craton-url>"
export OS_USERNAME="<username>"
export OS_PASSWORD="<password>"
export OS_PROJECT_ID="<project-id>"
```

And then `source` it into our environment:

```
$ source ~/cratonrc
# or
$ . ~/cratonrc
```

And finally we can use `craton` without those parameters.

Top-Level Options

Craton's command-line client has several top-level options. These are required to be specified prior to any sub-command. All of craton's top-level command-line options are documented here:

--version
Show the installed version of python-cratonclient.

Example usage:

```
$ craton --version
```

--format={default,json}
Specify the format of the output to the terminal. The default value is a pretty-printed table of information. Alternatively, users may request pretty-printed JSON.

Example usage:

```
$ craton --format=json host-list
$ craton --format=json region-show 1
```

--craton-url=URL
Specify the URL where Craton is reachable.

Example usage:

```
$ craton --craton-url=https://craton.cloud.corp host-list
```

--craton-version=VERSION
Control which version of Craton's API the client should use to communicate. At the moment, Craton only supports 1 for v1.

Example usage:

```
$ craton --craton-version=1 region-list
```

--os-project-id=OS_PROJECT_ID
Provide the Project ID to use when authenticating to Craton.

Example usage:

```
$ craton --os-project-id=b9f10eca66ac4c279c139d01e65f96b4 cell-list
```

--os-username=OS_USERNAME
Provide the Username to use when authenticating to Craton.

Example usage:

```
$ craton --os-username=demo project-list
```

--os-password=OS_PASWORD
Provide the Pasword to use when authenticating to Craton.

Example usage:

```
$ craton --os-password=demo devices-list
```

Subcommands

The craton command-line client has several subcommands. These include (but are not limited to):

- `help`
- `project-create`
- `project-delete`
- `project-list`
- `project-show`
- `cloud-create`
- `cloud-delete`
- `cloud-list`
- `cloud-show`
- `region-create`
- `region-delete`
- `region-list`
- `region-show`
- `cell-create`
- `cell-delete`
- `cell-list`
- `cell-show`
- `host-create`
- `host-delete`
- `host-list`
- `host-show`
- `device-list`.

The command-line options available for each command can be found via `craton help <subcommand-name>`, e.g.,

```
$ craton help cell-create
$ craton help host-list
```

CHAPTER 5

Python API User Guide

Once you have installed `python-cratonclient`, there are a few things you need to get started using the Python API:

1. You need to know how to authenticate to the Craton API you wish to talk to

   Some Craton API services will be deployed using Craton's in-built authentication system while others may use Keystone.

2. You need your credentials
3. You need the location of your Craton API service

This chapter of python-cratonclient's documentation is broken down into chapters:

Authenticating to Craton

There are two ways to authenticate to Craton:

1. Using Craton's in-built authentication system (the default)
2. Using Keystone

Craton Authentication

In the Craton Authentication case, you need the URL for the Craton API service, your username, project ID, and token. To set up cratonclient for this authentication, you need only do the following:

```
from cratonclient import auth
from cratonclient.v1 import client

craton_session = auth.craton_auth(
    username=USERNAME,
    token=TOKEN,
    project_id=PROJECT_ID,
```

```
)

craton = client.Client(
    session=craton_session,
    url=URL,
)
```

Keystone Authentication

When authenticating to Craton using Keystone, you need to know:

- the URL to use to authenticate to Keystone which we will refer to as `AUTH_URL`
- the username
- the password
- the project ID or name
- the user domain ID or name
- and the project domain ID or name

Then, we need to do the following:

```
from cratonclient import auth
from cratonclient.v1 import client

craton_session = auth.keystone_auth(
    auth_url=AUTH_URL,
    password=PASSWORD,
    username=USERNAME,
    user_domain_name=USER_DOMAIN_NAME,
    project_name=PROJECT_NAME,
    project_domain_name=PROJECT_DOMAIN_NAME,
)
craton = client.Client(
    session=craton_session,
    url=URL,
)
```

Communicating with Craton

Now that you've configured your authentication method, you can interact with your `craton` object like so:

```
for region in craton.regions.list():
    print('Region {} contains:'.format(region.name))
    for host in craton.hosts.list(region_id=region.id):
        print('    {}'.format(host.name))
```

The Craton API has the following resources:

- Cells
- Clouds
- Devices

- Hosts
- Network Devices
- Network Interfaces
- Networks
- Projects
- Regions
- Users

Of these:

- Cells
- Clouds
- Hosts
- Projects
- Regions

Are implemented.

Using the Clouds API

Here we will assume that we already have a `Client` instance configured with the appropriate authentication method (as demonstrated in *Authenticating to Craton*).

Listing Clouds

The Clouds API implements pagination. This means that by default, it does not return all clouds known to Craton. To ignore page limits and offsets, we can allow cratonclient to do handle pagination for us:

```
for cloud in craton.clouds.list():
    print_cloud_info(cloud)
```

By default *`list()`* will handle pagination for you. If, instead, you want to handle it yourself you will want to do something akin to:

```
first_page_of_clouds = list(craton.clouds.list(autopaginate=False))
marker_id = first_page_of_clouds[-1].id
second_page_of_clouds = list(craton.clouds.list(
    autopaginate=False,
    marker=marker_id,
))
marker_id = second_page_of_clouds[-1].id
third_page_of_clouds = list(craton.clouds.list(
    autopaginate=False,
    marker=marker_id,
))
# etc.
```

A more realistic example, however, might look like this:

```
clouds_list = None
marker = None
while clouds_list and clouds_list is not None:
    clouds_list = list(craton.clouds.list(
        marker=marker,
        autopaginate=False,
    ))
    # do something with clouds_list
    if clouds_list:
        marker = clouds_list[-1].id
```

This will have the effect of stopping the while loop when you eventually receive an empty list from `craton.clouds.list(...)`.

Creating Clouds

Clouds are the top-level item in Craton. To create a cloud, the only required item is a `name` for the cloud. This must be unique among clouds in the same project.

```
cloud = craton.clouds.create(
    name='my-cloud-0',
    note='This is my cloud, there are many like it, but this is mine.',
    variables={
        'some-var': 'some-var-value',
    },
)
```

Retrieving a Specific Cloud

Clouds can be retrieved by id.

```
cloud = craton.clouds.get(1)
```

Using a Cloud's Variables

Once we have a cloud we can introspect its variables like so:

```
cloud = craton.clouds.get(cloud_id)
cloud_vars = cloud.variables.get()
```

To update them:

```
updated_vars = {
    'var-a': 'new-var-a',
    'var-b': 'new-var-b',
    'updated-var': 'updated value',
}
cloud.variables.update(**updated_vars)
```

To delete them:

```
cloud.variables.delete('var-a', 'var-b', 'updated-var')
```

Updating a Cloud

We can update a cloud's attributes (but not its variables) like so:

```
craton.clouds.update(
    cloud_id,
    name='new name',
    note='Updated note.',
)
```

Most attributes that you can specify on creation can also be specified for updating the cloud as well.

Deleting a Cloud

We can delete with only its id:

```
craton.clouds.delete(cloud_id)
```

Using the Regions API

Here we will assume that we already have a `Client` instance configured with the appropriate authentication method (as demonstrated in *Authenticating to Craton*).

Listing Regions

The Regions API implements pagination. This means that by default, it does not return all regions known to Craton. To ignore page limits and offsets, we can allow cratonclient to do handle pagination for us:

```
for region in craton.regions.list():
    print_region_info(region)
```

By default *`list()`* will handle pagination for you. If, instead, you want to handle it yourself you will want to do something akin to:

```
first_page_of_regions = list(craton.regions.list(autopaginate=False))
marker_id = first_page_of_regions[-1].id
second_page_of_regions = list(craton.regions.list(
    autopaginate=False,
    marker=marker_id,
))
marker_id = second_page_of_regions[-1].id
third_page_of_regions = list(craton.regions.list(
    autopaginate=False,
    marker=marker_id,
))
# etc.
```

A more realistic example, however, might look like this:

```
regions_list = None
marker = None
while regions_list and regions_list is not None:
    regions_list = list(craton.regions.list(
```

```
        marker=marker,
        autopaginate=False,
    ))
    # do something with regions_list
    if regions_list:
        marker = regions_list[-1].id
```

This will have the effect of stopping the while loop when you eventually receive an empty list from `craton.regions.list(...)`.

Creating Regions

Regions are required to be part of a Cloud in Craton. To create a region, the only required items are a `name` for the region and the ID of the cloud it belongs to. The name must be unique among regions in the same project.

```
region = craton.regions.create(
    name='my-region-0',
    cloud_id=cloud_id,
    note='This is my region, there are many like it, but this is mine.',
    variables={
        'some-var': 'some-var-value',
    },
)
```

Retrieving a Specific Region

Regions can be retrieved by id.

```
region = craton.regions.get(1)
```

Using a Region's Variables

Once we have a region we can introspect its variables like so:

```
region = craton.regions.get(region_id)
region_vars = region.variables.get()
```

To update them:

```
updated_vars = {
    'var-a': 'new-var-a',
    'var-b': 'new-var-b',
    'updated-var': 'updated value',
}
region.variables.update(**updated_vars)
```

To delete them:

```
region.variables.delete('var-a', 'var-b', 'updated-var')
```

Updating a Region

We can update a region's attributes (but not its variables) like so:

```
craton.regions.update(
    region_id,
    name='new name',
    note='Updated note.',
)
```

Most attributes that you can specify on creation can also be specified for updating the region as well.

Deleting a Region

We can delete with only its id:

```
craton.regions.delete(region_id)
```

Using the Cells API

Here we will assume that we already have a `Client` instance configured with the appropriate authentication method (as demonstrated in *Authenticating to Craton*).

Listing Cells

The Cells API implements pagination. This means that by default, it does not return all cells known to Craton. To ignore page limits and offsets, we can allow cratonclient to do handle pagination for us:

```
for cell in craton.cells.list():
    print_cell_info(cell)
```

By default *`list()`* will handle pagination for you. If, instead, you want to handle it yourself you will want to do something akin to:

```
first_page_of_cells = list(craton.cells.list(autopaginate=False))
marker_id = first_page_of_cells[-1].id
second_page_of_cells = list(craton.cells.list(
    autopaginate=False,
    marker=marker_id,
))
marker_id = second_page_of_cells[-1].id
third_page_of_cells = list(craton.cells.list(
    autopaginate=False,
    marker=marker_id,
))
# etc.
```

A more realistic example, however, might look like this:

```
cells_list = None
marker = None
while cells_list and cells_list is not None:
    cells_list = list(craton.cells.list(
```

```
        marker=marker,
        autopaginate=False,
    ))
    # do something with cells_list
    if cells_list:
        marker = cells_list[-1].id
```

This will have the effect of stopping the while loop when you eventually receive an empty list from `craton.cells.list(...)`.

Creating Cells

Cells live below a Region in Craton. To create a cell, the only required items are a `name` for the cell, a cloud ID, and a region ID. The name must be unique among cells in the same project.

```
cell = craton.cells.create(
    name='my-cell-0',
    cloud_id=cloud_id,
    region_id=region_id,
    note='This is my cell, there are many like it, but this is mine.',
    variables={
        'some-var': 'some-var-value',
    },
)
```

Retrieving a Specific Cell

Cells can be retrieved by id.

```
cell = craton.cells.get(1)
```

Using a Cell's Variables

Once we have a cell we can introspect its variables like so:

```
cell = craton.cells.get(cell_id)
cell_vars = cell.variables.get()
```

To update them:

```
updated_vars = {
    'var-a': 'new-var-a',
    'var-b': 'new-var-b',
    'updated-var': 'updated value',
}
cell.variables.update(**updated_vars)
```

To delete them:

```
cell.variables.delete('var-a', 'var-b', 'updated-var')
```

Updating a Cell

We can update a cell's attributes (but not its variables) like so:

```
craton.cells.update(
    cell_id,
    name='new name',
    note='Updated note.',
)
```

Most attributes that you can specify on creation can also be specified for updating the cell as well.

Deleting a Cell

We can delete with only its id:

```
craton.cells.delete(cell_id)
```

Using the Hosts API

Here we will assume that we already have a `Client` instance configured with the appropriate authentication method (as demonstrated in *Authenticating to Craton*).

Listing Hosts

The Hosts API implements pagination. This means that by default, it does not return all hosts known to Craton. To ignore page limits and offsets, we can allow cratonclient to do handle pagination for us:

```
for host in craton.hosts.list():
    print_host_info(host)
```

By default *`list()`* will handle pagination for you. If, instead, you want to handle it yourself you will want to do something akin to:

```
first_page_of_hosts = list(craton.hosts.list(autopaginate=False))
marker_id = first_page_of_hosts[-1].id
second_page_of_hosts = list(craton.hosts.list(
    autopaginate=False,
    marker=marker_id,
))
marker_id = second_page_of_hosts[-1].id
third_page_of_hosts = list(craton.hosts.list(
    autopaginate=False,
    marker=marker_id,
))
# etc.
```

A more realistic example, however, might look like this:

```
hosts_list = None
marker = None
while hosts_list and hosts_list is not None:
    hosts_list = list(craton.hosts.list(
```

```
        marker=marker,
        autopaginate=False,
    ))
    # do something with hosts_list
    if hosts_list:
        marker = hosts_list[-1].id
```

This will have the effect of stopping the while loop when you eventually receive an empty list from `craton.hosts.list(...)`.

Creating Hosts

Hosts live inside either a Region or Cell in Craton. To create a host, one needs:

- A unique name
- A unique IP address
- A "device type" (this is freeform), e.g., "server", "container", "nova-vm", etc.
- A cloud ID
- A region ID

```
host = craton.hosts.create(
    name='my-host-0',
    ip_address='127.0.1.0',
    device_type='server',
    cloud_id=cloud_id,
    region_id=region_id,
    note='This is my host, there are many like it, but this is mine.',
    variables={
        'some-var': 'some-var-value',
    },
)
```

Retrieving a Specific Host

Hosts can be retrieved by id.

```
host = craton.hosts.get(1)
```

Using a Host's Variables

Once we have a host we can introspect its variables like so:

```
host = craton.hosts.get(host_id)
host_vars = host.variables.get()
```

To update them:

```
updated_vars = {
    'var-a': 'new-var-a',
    'var-b': 'new-var-b',
    'updated-var': 'updated value',
```

```
}
host.variables.update(**updated_vars)
```

To delete them:

```
host.variables.delete('var-a', 'var-b', 'updated-var')
```

Updating a Host

We can update a host's attributes (but not its variables) like so:

```
craton.hosts.update(
    host_id,
    name='new name',
    note='Updated note.',
)
```

Most attributes that you can specify on creation can also be specified for updating the host as well.

Deleting a Host

We can delete with only its id:

```
craton.hosts.delete(host_id)
```

Using the Devices API

Here we will assume that we already have a `Client` instance configured with the appropriate authentication method (as demonstrated in *Authenticating to Craton*).

Note: The Devices API is quite unlike other API endpoints presently in craton. At the moment, it returns both hosts and network devices. It concatenates the two lists in an indeterminate manner. On one invocation, you may receive the hosts first and then the network devices, on another you may receive them in the alternate order. If more items are returned in these listings, then the number of different orderings will only increase factorially.

Listing Devices

The Devices API implements pagination. This means that by default, it does not return all devices known to Craton. To ignore page limits and offsets, we can allow cratonclient to do handle pagination for us:

```
for device in craton.devices.list():
    print_device_info(device)
```

By default *`list()`* will handle pagination for you.

Using the Projects API

Here we will assume that we already have a `Client` instance configured with the appropriate authentication method (as demonstrated in *Authenticating to Craton*).

Listing Projects

The Projects API implements pagination. This means that by default, it does not return all projects known to Craton. To ignore page limits and offsets, we can allow cratonclient to do handle pagination for us:

```
for project in craton.projects.list():
    print_project_info(project)
```

By default *`list()`* will handle pagination for you. If, instead, you want to handle it yourself you will want to do something akin to:

```
first_page_of_projects = list(craton.projects.list(autopaginate=False))
marker_id = first_page_of_projects[-1].id
second_page_of_projects = list(craton.projects.list(
    autopaginate=False,
    marker=marker_id,
))
marker_id = second_page_of_projects[-1].id
third_page_of_projects = list(craton.projects.list(
    autopaginate=False,
    marker=marker_id,
))
# etc.
```

A more realistic example, however, might look like this:

```
projects_list = None
marker = None
while projects_list and projects_list is not None:
    projects_list = list(craton.projects.list(
        marker=marker,
        autopaginate=False,
    ))
    # do something with projects_list
    if projects_list:
        marker = projects_list[-1].id
```

This will have the effect of stopping the while loop when you eventually receive an empty list from `craton.projects.list(...)`.

Creating Projects

Projects are top-level items in Craton. To create a project, one needs:

- A unique name
- Permission to create new projects

```
project = craton.projects.create(
    name='my-project-0',
```

```
    variables={
        'some-var': 'some-var-value',
    },
)
```

Retrieving a Specific Project

Projects can be retrieved by id.

```
project = craton.projects.get(1)
```

Using a Project's Variables

Once we have a project we can introspect its variables like so:

```
project = craton.projects.get(project_id)
project_vars = project.variables.get()
```

To update them:

```
updated_vars = {
    'var-a': 'new-var-a',
    'var-b': 'new-var-b',
    'updated-var': 'updated value',
}
project.variables.update(**updated_vars)
```

To delete them:

```
project.variables.delete('var-a', 'var-b', 'updated-var')
```

Updating a Project

We can update a project's attributes (but not its variables) like so:

```
craton.projects.update(
    project_id,
    name='new name',
)
```

Most attributes that you can specify on creation can also be specified for updating the project as well.

Deleting a Project

We can delete with only its id:

```
craton.projects.delete(project_id)
```

CHAPTER 6

Python API Reference Documentation

This chapter of python-cratonclient's documentation focuses entirely on the API of the different objects involved in the use of cratonclient's Python API.

Version-less Objects

class `cratonclient.session.`**`Session`**(*session=None*, *username=None*, *token=None*, *project_id=None*)

Management class to allow different types of sessions to be used.

If an instance of Craton is deployed with Keystone Middleware, this allows for a keystoneauth session to be used so authentication will happen immediately.

v1 API Documentation

class `cratonclient.v1.client.`**`Client`**(*session*, *url*)

Craton v1 API Client.

`cells`

The canonical way to list, get, delete, or update cell objects via a *`CellManager`* instance.

`clouds`

The canonical way to list, get, delete, or update cloud objects via a *`CloudManager`* instance.

`devices`

The canonical way to list devicess via a *`DeviceManager`* instance.

`hosts`

The canonical way to list, get, delete, or update host objects via a *`HostManager`* instance.

`projects`

The canonical way to list, get, delete, or update project objects via a *`ProjectManager`* instance.

regions
The canonical way to list, get, delete, or update region objects via a *RegionManager* instance.

Cells

class cratonclient.v1.cells.**Cell**(*manager*, *info*, *loaded=False*)
Representation of a Region.

delete()
Delete the resource from the service.

get()
Support for lazy loading details.

Some clients, such as novaclient have the option to lazy load the details, details which can be loaded with this function.

human_id
Human-readable ID which can be used for bash completion.

is_loaded()
Check if the resource has been loaded.

class cratonclient.v1.cells.**CellManager**(*session*, *url*, ***extra_request_kwargs*)
A manager for cells.

create(*skip_merge=False*, ***kwargs*)
Create a new item based on the keyword arguments provided.

delete(*item_id=None*, *skip_merge=True*, *json=None*, ***kwargs*)
Delete the item based on the keyword arguments provided.

get(*item_id=None*, *skip_merge=True*, ***kwargs*)
Retrieve the item based on the keyword arguments provided.

list(*skip_merge=False*, ***kwargs*)
Generate the items from this endpoint.

update(*item_id=None*, *skip_merge=True*, ***kwargs*)
Update the item based on the keyword arguments provided.

Clouds

class cratonclient.v1.clouds.**Cloud**(*manager*, *info*, *loaded=False*)
Representation of a Cloud.

delete()
Delete the resource from the service.

get()
Support for lazy loading details.

Some clients, such as novaclient have the option to lazy load the details, details which can be loaded with this function.

human_id
Human-readable ID which can be used for bash completion.

is_loaded()
Check if the resource has been loaded.

class cratonclient.v1.clouds.**CloudManager**(*session*, *url*, ***extra_request_kwargs*)
A manager for clouds.

create(*skip_merge=False*, ***kwargs*)
Create a new item based on the keyword arguments provided.

delete(*item_id=None*, *skip_merge=True*, *json=None*, ***kwargs*)
Delete the item based on the keyword arguments provided.

get(*item_id=None*, *skip_merge=True*, ***kwargs*)
Retrieve the item based on the keyword arguments provided.

list(*skip_merge=False*, ***kwargs*)
Generate the items from this endpoint.

update(*item_id=None*, *skip_merge=True*, ***kwargs*)
Update the item based on the keyword arguments provided.

Devices

class cratonclient.v1.devices.**Device**(*manager*, *info*, *loaded=False*)
Representation of a Device.

delete()
Delete the resource from the service.

get()
Support for lazy loading details.

Some clients, such as novaclient have the option to lazy load the details, details which can be loaded with this function.

human_id
Human-readable ID which can be used for bash completion.

is_loaded()
Check if the resource has been loaded.

class cratonclient.v1.devices.**DeviceManager**(*session*, *url*, ***extra_request_kwargs*)
A manager for devices.

list(***kwargs*)
Generate the items from this endpoint.

Hosts

class cratonclient.v1.hosts.**Host**(*manager*, *info*, *loaded=False*)
Representation of a Host.

delete()
Delete the resource from the service.

get()
Support for lazy loading details.

Some clients, such as novaclient have the option to lazy load the details, details which can be loaded with this function.

human_id
Human-readable ID which can be used for bash completion.

is_loaded()
Check if the resource has been loaded.

class cratonclient.v1.hosts.**HostManager**(*session*, *url*, ***extra_request_kwargs*)
A manager for hosts.

create(*skip_merge=False*, ***kwargs*)
Create a new item based on the keyword arguments provided.

delete(*item_id=None*, *skip_merge=True*, *json=None*, ***kwargs*)
Delete the item based on the keyword arguments provided.

get(*item_id=None*, *skip_merge=True*, ***kwargs*)
Retrieve the item based on the keyword arguments provided.

list(*skip_merge=False*, ***kwargs*)
Generate the items from this endpoint.

update(*item_id=None*, *skip_merge=True*, ***kwargs*)
Update the item based on the keyword arguments provided.

Projects

class cratonclient.v1.projects.**Project**(*manager*, *info*, *loaded=False*)
Representation of a Project.

delete()
Delete the resource from the service.

get()
Support for lazy loading details.

Some clients, such as novaclient have the option to lazy load the details, details which can be loaded with this function.

human_id
Human-readable ID which can be used for bash completion.

is_loaded()
Check if the resource has been loaded.

class cratonclient.v1.projects.**ProjectManager**(*session*, *url*, ***extra_request_kwargs*)
A manager for projects.

create(*skip_merge=False*, ***kwargs*)
Create a new item based on the keyword arguments provided.

delete(*item_id=None*, *skip_merge=True*, *json=None*, ***kwargs*)
Delete the item based on the keyword arguments provided.

get(*item_id=None*, *skip_merge=True*, ***kwargs*)
Retrieve the item based on the keyword arguments provided.

list(*skip_merge=False*, ***kwargs*)
Generate the items from this endpoint.

update(*item_id=None*, *skip_merge=True*, ***kwargs*)
Update the item based on the keyword arguments provided.

Regions

class cratonclient.v1.regions.**Region**(*manager*, *info*, *loaded=False*)
Representation of a Region.

delete()
Delete the resource from the service.

get()
Support for lazy loading details.

Some clients, such as novaclient have the option to lazy load the details, details which can be loaded with this function.

human_id
Human-readable ID which can be used for bash completion.

is_loaded()
Check if the resource has been loaded.

class cratonclient.v1.regions.**RegionManager**(*session*, *url*, ***extra_request_kwargs*)
A manager for regions.

create(*skip_merge=False*, ***kwargs*)
Create a new item based on the keyword arguments provided.

delete(*item_id=None*, *skip_merge=True*, *json=None*, ***kwargs*)
Delete the item based on the keyword arguments provided.

get(*item_id=None*, *skip_merge=True*, ***kwargs*)
Retrieve the item based on the keyword arguments provided.

list(*skip_merge=False*, ***kwargs*)
Generate the items from this endpoint.

update(*item_id=None*, *skip_merge=True*, ***kwargs*)
Update the item based on the keyword arguments provided.

Variables

class cratonclient.v1.variables.**Variable**(*name*, *value*)
Represents a Craton variable key/value pair.

class cratonclient.v1.variables.**Variables**(*manager*, *info*, *loaded=False*)
Represents a dictionary of Variables.

class cratonclient.v1.variables.**VariableManager**(*session*, *url*, ***extra_request_kwargs*)
A CRUD manager for variables.

create(*skip_merge=False*, ***kwargs*)
Create a new item based on the keyword arguments provided.

delete(**args*, ***kwargs*)
Wrap crud.CRUDClient's delete to simplify for the variables.

One can pass in a series of keys to delete, and this will pass the correct arguments to the crud.CRUDClient.delete function.

```
>>> craton.hosts.get(1234).variables.delete('var-a', 'var-b')
<Response [204]>
```

get (*item_id=None*, *skip_merge=True*, ***kwargs*)
Retrieve the item based on the keyword arguments provided.

list (*skip_merge=False*, ***kwargs*)
Generate the items from this endpoint.

update (*item_id=None*, *skip_merge=True*, ***kwargs*)
Update the item based on the keyword arguments provided.

Authentication Helpers

cratonclient.auth.**craton_auth** (*username*, *token*, *project_id*, *verify=True*)
Configure a cratonclient Session to authenticate to Craton.

This will create, configure, and return a Session object that will use Craton's built-in authentication method.

Parameters

- **username** (*str*) – The username with which to authentiate against the API.
- **token** (*str*) – The token with which to authenticate against the API.
- **project_id** (*str*) – The project ID that the user belongs to.
- **verify** (*bool*) – (Optional) Whether or not to verify HTTPS certificates provided by the server. Default: True

Returns Configured cratonclient session.

Return type *cratonclient.session.Session*

Example:

```
from cratonclient import auth
from cratonclient.v1 import client

craton = client.Client(session=auth.craton_auth(
    username='demo',
    token='demo',
    project_id='b9f10eca66ac4c279c139d01e65f96b4',
))
```

cratonclient.auth.**keystone_auth** (*auth_url*, *username*, *password*, *verify=True*, *project_name=None*, *project_id=None*, *project_domain_name=None*, *project_domain_id=None*, *user_domain_name=None*, *user_domain_id=None*, ***auth_parameters*)
Configure a cratonclient Session to authenticate with Keystone.

This will create, configure, and return a Session using thet appropriate Keystone authentication plugin to be able to communicate and authenticate to Craton.

Note: Presently, this function supports only V3 Password based authentication to Keystone. We also do not validate that you specify required attributes. For example, Keystone will require you provide `project_name` or `project_id` but we will not enforce whether or not you've specified one.

Parameters

- **auth_url** (*str*) – The URL of the Keystone instance to authenticate to.
- **username** (*str*) – The username with which we will authenticate to Keystone.
- **password** (*str*) – The password used to authenticate to Keystone.
- **project_name** (*str*) – (Optional) The name of the project the user belongs to.
- **project_id** (*str*) – (Optional) The ID of the project the user belongs to.
- **project_domain_name** (*str*) – (Optional) The name of the project's domain.
- **project_domain_id** (*str*) – (Optional) The ID of the project's domain.
- **user_domain_name** (*str*) – (Optional) The name of the user's domain.
- **user_domain_id** (*str*) – (Optional) The ID of the user's domain.
- **verify** (*bool*) – (Optional) Whether or not to verify HTTPS certificates provided by the server. Default: True
- ****auth_parameters** – Any extra authentication parameters used to authenticate to Keystone. See the Keystone documentation for usage of: - `trust_id` - `domain_id` - `domain_name` - `reauthenticate`

Returns Configured cratonclient session.

Return type *cratonclient.session.Session*

Example:

```
from cratonclient import auth
from cratonclient.v1 import client

craton = client.Client(session=auth.keystone_auth(
    auth_url='https://keystone.cloud.org/v3',
    username='admin',
    password='s3cr373p@55w0rd',
    project_name='admin',
    project_domain_name='Default',
    user_domain_name='Default',
))
```

Specifications for python-cratonclient:

CHAPTER 7

Fleet Management Service Specifications

All current approved Craton API specifications:

Testing Plan for cratonclient

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/python-cratonclient/+spec/testing-plan

The python-cratonclient presently has a couple hundred tests that ensure that it does what we expect. At the moment, however, most of those tests use mocks to force certain behaviours. While these are valuable tests, we need more real world tests to ensure that the client *does* reflect the reality of the API.

Problem description

Using Mocks inside tests can be a very valuable tool for writing fast tests and reproducing issues that are otherwise difficult to do. Mocks, however, can make tests incredibly fragile. To have a healthy test suite, we need tests that avoid mocks altogether.

Proposed change

The existing test suite is a good base to build a better test suite upon. Going forward, we will *still* rely on Mock for *some* tests but we will instead start to rely on other test methodologies for this client.

First we will implement a new type of integration testing. This will leverage a library called Betamax. This will allow us to have tests that use real data and real responses from a Craton server without needing to set a server up every single time. Betamax will record the requests and responses and allow our tests to use the saved interactions (request, response pairs) instead of requiring an active server to be configured.

Next we'll move our existing integration tests that heavily mock out the layers below into the unit test directories unless they actually test some level of integration (i.e., more than one level).

Finally we'll add the ability to run tests against a live server that will be configured and managed by docker-py in a way similar to the Craton API functional tests. These will be part of a separate tox test environment, similar to the API tests.

Alternatives

Much of the client relies on responses from the API. We could simply require functional tests for every layer of the client library that interacts with the API which would leave only utility functions for unit testing. This would reduce the need for different tests at different levels. This would also increase the run time of the default test suite by requiring docker containers be launched for each one.

Security impact

If any impact, this may improve the posture by increasing the testing of the user's surface area.

Other end user impact

None

Performance Impact

None

Developer impact

This should be accompanied by developer documentation with information determining what kind of test to write when and what tests belong in what categories.

Implementation

Assignee(s)

Primary assignee: - icordasc

Other contributors: - None

Work Items

- Add base framework for testing with Betamax.
- Add comprehensive integration tests for the Python API level.
- Move existing "integration" unit tests into appropriate unit test modules.
- Improve shell integration testing. Including parsing PrettyTable output and using the JSON formatter for better verification of reporting.
- Add functional tests against live Craton servers.

Dependencies

- None

Testing

See above. =)

Documentation Impact

This will require developer reference updates as noted above.

References

N/A

All implemented Craton API specifications:

Indices and Tables

- search

CHAPTER 8

Indices and tables

- genindex
- search

Index

Symbols

C

D

G