Chameleon Cloud Python API

Release 0.1

Nick Timkovich

Oct 26, 2021

MODULES

`python-chi` is a Python library that can help you interact with the Chameleon testbed to improve your workflows with automation. It additionally pairs well with environments like Jupyter Notebooks.

CHAPTER ONE

INSTALLATION

```
pip install python-chi
```

CHAPTER

TWO

AUTHENTICATION

Environment variables are the primary authentication method. Please refer to the documentation on OpenRC scripts to learn more about how to download and source your authentication credentials for the CLI; the same instructions apply for using the Python interface.

CHAPTER

THREE

BASIC USAGE

The following example shows how to make a reservation for a bare metal server. For more details about the modules available refer to their respective pages.

```
import chi

# Select your project
chi.set('project_name', 'CH-XXXXXX')
# Select your site
chi.use_site('CHI@UC')

# Make a reservation ...
reservations = []
# ... for one node of type "compute_skylake"
chi.lease.add_node_reservation(
    reservations, node_type='compute_skylake', count=1)
# ... and one Floating IP
chi.lease.add_fip_reservation(count=1)
# ... for one day.
start_date, end_date = chi.lease.lease_duration(days=1)
chi.lease.create_lease(
    lease_name, reservations, start_date=start_date, end_date=end_date)
```

3.1 chi

The *chi* module exposes a global context object and helpers for generating clients for interfacing with the various Chameleon services.

`chi.`**`blazar`**(*session=None*) → BlazarClient

Get a preconfigured client for Blazar, the reservation service.

Parameters **`session`** (*`Session`*) – An authentication session object. By default a new session is created via *`chi.session()`*.

Returns A new Blazar client.

`chi.`**`connection`**(*session=None*) → Connection

Get connection context for OpenStack SDK.

The returned `openstack.connection.Connection` object has several proxy modules attached for each service provided by the cloud.

Note: For the most part, it is more straightforward to use clients specific to the service you are targeting. However, some of the proxy modules are useful for operations that span a few services, such as assigning a Floating IP to a server instance.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via *chi.session()*.

Returns A new connection proxy.

chi.**get**(*key*)

Get a context parameter by name.

Parameters **key** (*str*) – the parameter name.

Returns the parameter value.

Return type any

Raises **cfg.NoSuchOptError** – if the parameter is not supported.

chi.**glance**(*session=None*) → GlanceClient

Get a preconfigured client for Glance, the image service.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via *chi.session()*.

Returns A new Glance client.

chi.**gnocchi**(*session=None*) → GnocchiClient

Get a preconfigured client for Gnocchi, the metrics service.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via *chi.session()*.

Returns A new Gnocchi client.

chi.**ironic**(*session=None*) → IronicClient

Get a preconfigured client for Ironic, the bare metal service.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via *chi.session()*.

Returns A new Ironic client.

chi.**keystone**(*session=None*) → KeystoneClient

Get a preconfigured client for Keystone, the authentication service.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via *chi.session()*.

Returns A new Keystone client.

chi.**neutron**(*session=None*) → NeutronClient

Get a preconfigured client for Neutron, the networking service.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via *chi.session()*.

Returns A new Neutron client.

chi.**nova**(*session=None*) → NovaClient

Get a preconfigured client for Nova, the compute service.

Parameters **session** (*Session*) – An authentication session object. By default a new session is created via `chi.session()`.

Returns A new Nova client.

`chi.params()`

List all parameters currently set on the context.

Returns a list of parameter names.

Return type List[str]

`chi.reset()`

Reset the context, removing all overrides and defaults.

The `auth_type` parameter will be defaulted to the value of the OS_AUTH_TYPE environment variable, falling back to "v3token" if not defined.

All context parameters will revert to the default values inferred from environment variables.

`chi.session()`

Get a Keystone Session object suitable for authenticating a client.

Returns the authentication session object.

Return type keystoneauth1.session.Session

`chi.set(key, value)`

Set a context parameter by name.

Parameters

- **key** (*str*) – the parameter name.
- **value** (*any*) – the parameter value.

Raises **cfg.NoSuchOptError** – if the parameter is not supported.

`chi.use_site(site_name)`

Configure the global request context to target a particular CHI site.

Targeting a site will mean that leases, instance launch requests, and any other API calls will be sent to that site. By default, no site is selected, and one must be explicitly chosen.

```
chi.use_site("CHI@UC")
```

Changing the site will affect future calls the client makes, implicitly. Therefore something like this is possible:

```
chi.use_site("CHI@UC")
chi.lease.create_lease("my-uc-lease", reservations)
chi.use_site("CHI@TACC")
chi.lease.create_lease("my-tacc-lease", reservations)
```

Parameters **site_name** (*str*) – The name of the site, e.g., "CHI@UC".

3.2 chi.lease

The `chi.lease` module exposes both a functional interface and an object-oriented interface for interacting with resource leases.

3.2.1 Functional interface

`chi.lease.`**`add_device_reservation`**(*reservation_list*, *count=1*, *device_model=None*, *device_name=None*)

Add an IoT/edge device reservation to a reservation list.

Parameters

- **`reservation_list`** (*`list[dict]`*) – The list of reservations to add to.
- **`count`** (*`int`*) – The number of devices to request.
- **`device_model`** (*`str`*) – The model of device to reserve. This should match a "model" property of the devices registered in Blazar.
- **`device_name`** (*`str`*) – The name of a specific device to reserve. If this is provided in conjunction with `count` or other constraints, an error will be raised, as there is only 1 possible device that can match this criteria, because devices have unique names.

Raises `ValueError` – If `device_name` is provided, but `count` is greater than 1, or some other constraint is present.

`chi.lease.`**`add_fip_reservation`**(*reservation_list*, *count=1*)

Add a floating IP reservation to a reservation list.

Parameters

- **`reservation_list`** (*`list[dict]`*) – The list of reservations to add to. The list will be extended in-place.
- **`count`** (*`int`*) – The number of floating IPs to reserve.

`chi.lease.`**`add_network_reservation`**(*reservation_list*, *network_name*, *of_controller_ip=None*, *of_controller_port=None*, *vswitch_name=None*, *physical_network='physnet1'*)

Add a network reservation to a reservation list.

Parameters

- **`reservation_list`** (*`list[dict]`*) – The list of reservations to add to. The list will be extended in-place.
- **`network_name`** (*`str`*) – The name of the network to create when the reservation starts.
- **`of_controller_ip`** (*`str`*) – The OpenFlow controller IP, if the network should be controlled by an external controller.
- **`of_controller_port`** (*`int`*) – The OpenFlow controller port.
- **`vswitch_name`** (*`str`*) – The name of the virtual switch associated with this network. See the virtual forwarding context documentation for more details.
- **`physical_network`** (*`str`*) – The physical provider network to reserve from. This only needs to be changed if you are reserving a stitchable network. (Default "physnet1").

`chi.lease.`**`add_node_reservation`**(*reservation_list*, *count=1*, *resource_properties=[]*, *node_type=None*, *architecture=None*)

Add a node reservation to a reservation list.

Parameters

- **reservation_list** (*list[dict]*) – The list of reservations to add to. The list will be extended in-place.
- **count** (*int*) – The number of nodes of the given type to request. (Default 1).
- **resource_properties** (*list*) – A list of resource property constraints. These take the form [<operation>, <search_key>, <search_value>], e.g.:

```
["==", "$node_type", "some-node-type"]: filter the reservation to␣
↪only
  nodes with a `node_type` matching "some-node-type".
[">", "$architecture.smt_size", 40]: filter to nodes having more␣
↪than 40
  (hyperthread) cores.
```

- **node_type** (*str*) – The node type to request. If None, the reservation will not target any particular node type. If *resource_properties* is defined, the node type constraint is added to the existing property constraints.
- **architecture** (*str*) – The node architecture to request. If *resource_properties* is defined, the architecture constraint is added to the existing property constraints.

`chi.lease.`**`create_lease`**(*lease_name*, *reservations=[]*, *start_date=None*, *end_date=None*)

Create a new lease with some requested reservations.

Parameters

- **lease_name** (*str*) – The name to give the new lease.
- **reservations** (*list[dict]*) – The reservations to request with the lease.
- **start_date** (*datetime*) – The start date of the lease. (Defaults to now.)
- **end_date** (*datetime*) – The end date of the lease. (Defaults to 1 day from the lease start date.)

Returns The created lease representation.

`chi.lease.`**`delete_lease`**(*ref*)

Delete the lease.

Parameters **ref** (*str*) – The name or ID of the lease.

`chi.lease.`**`get_device_reservation`**(*lease_ref*, *count=None*, *device_model=None*, *device_name=None*)

Retrieve a reservation ID for a device reservation.

The reservation ID is useful to have when requesting containers.

Parameters

- **lease_ref** (*str*) – The ID or name of the lease.
- **count** (*int*) – An optional count of devices the desired reservation was made for. Use this if you have multiple reservations under a lease.
- **device_model** (*str*) – An optional device model the desired reservation was made for. Use this if you have multiple reservations under a lease.
- **device_name** (*str*) – An optional device name the desired reservation was made for. Use this if you have multiple reservations under a lease.

Returns The ID of the reservation, if found.

Raises `ValueError` – If no reservation was found, or multiple were found.

`chi.lease.`**`get_lease`**`(`*ref*`)` → dict

Get a lease by its ID or name.

Parameters **`ref`** (*`str`*) – The ID or name of the lease.

Returns The lease matching the ID or name.

`chi.lease.`**`get_lease_id`**`(`*lease_name*`)` → str

Look up a lease's ID from its name.

Parameters **`name`** (*`str`*) – The name of the lease.

Returns The ID of the found lease.

Raises `ValueError` – If the lease could not be found, or if multiple leases were found with the same name.

`chi.lease.`**`get_node_reservation`**`(`*lease_ref*, *count=None*, *resource_properties=None*, *node_type=None*, *architecture=None*`)`

Retrieve a reservation ID for a node reservation.

The reservation ID is useful to have when launching bare metal instances.

Parameters

- **`lease_ref`** (*`str`*) – The ID or name of the lease.
- **`count`** (*`int`*) – An optional count of nodes the desired reservation was made for. Use this if you have multiple reservations under a lease.
- **`resource_properties`** (*`list`*) – An optional set of resource property constraints the desired reservation was made under. Use this if you have multiple reservations under a lease.
- **`node_type`** (*`str`*) – An optional node type the desired reservation was made for. Use this if you have multiple reservations under a lease.
- **`architecture`** (*`str`*) – An optional node architecture the desired reservation was made for. Use this if you have multiple reservations under a lease.

Returns The ID of the reservation, if found.

Raises `ValueError` – If no reservation was found, or multiple were found.

`chi.lease.`**`get_reserved_floating_ips`**`(`*lease_ref*`)` → list[str]

Get a list of Floating IP addresses reserved in a lease.

Parameters **`lease_ref`** (*`str`*) – The ID or name of the lease.

Returns A list of all reserved Floating IP addresses, if any were reserved.

`chi.lease.`**`lease_duration`**`(`*days=1*, *hours=0*`)`

Compute the start and end dates for a lease given its desired duration.

When providing both `days` and `hours`, the duration is summed. So, the following would be a lease for one and a half days:

```
start_date, end_date = lease_duration(days=1, hours=12)
```

Parameters

- **`days`** (*`int`*) – The number of days the lease should be for.
- **`hours`** (*`int`*) – The number of hours the lease should be for.

`chi.lease.`**`wait_for_active`**`(`*ref*`)`

Wait for the lease to become active.

This function will wait for 2.5 minutes, which is a somewhat arbitrary amount of time.

Parameters **`ref`** (*`str`*) – The name or ID of the lease.

Returns The lease in ACTIVE state.

Raises **`TimeoutError`** – If the lease fails to become active within the timeout.

3.2.2 Object-oriented interface

`class` `chi.lease.`**`Lease`**`(`***kwargs*`)`

Creates and manages a lease, optionally with a context manager (`with`).

```
with Lease(session, node_type='compute_skylake') as lease:
    instance = lease.create_server()
    ...
```

When using the context manager, on entering it will wait for the lease to launch, then on exiting it will delete the lease, which in-turn also deletes the instances launched with it.

Parameters

- **`keystone_session`** – session object
- **`sequester`** (*`bool`*) – If the context manager catches that an instance failed to start, it will not delete the lease, but rather extend it and rename it with the ID of the instance that failed.
- **`_no_clean`** (*`bool`*) – Don't delete the lease at the end of a context manager
- **`kwargs`** – Parameters passed through to `lease_create_nodetype()` and in turn `lease_create_args()`

`create_server``(`**server_args*, ***server_kwargs*`)`

Generates instances using the resource of the lease. Arguments are passed to `ccmanage.server.Server` and returns same object.

`delete``()`

Deletes the lease

`classmethod from_existing``(`*id*`)`

Attach to an existing lease by ID. When using in conjunction with the context manager, it will *not* delete the lease at the end.

`property ready`

Returns True if the lease has started.

`refresh``()`

Updates the lease data

`property status`

Refreshes and returns the status of the lease.

`wait``()`

Blocks for up to 150 seconds, waiting for the lease to be ready. Raises a RuntimeError if it times out.

3.3 chi.server

The *chi.server* module exposes both a functional interface and an object-oriented interface for interacting with server instances.

3.3.1 Functional interface

Any of the following functions can be directly imported and used individually:

```
from chi.server import get_server

s = server.get_server('my-server-name')
```

`chi.server.`**`associate_floating_ip`**(*server_id*, *floating_ip_address=None*)

Associate an allocated Floating IP with a server.

If no Floating IP is specified, one will be allocated dynamically.

Parameters

- **`server_id`** (*str*) – The ID of the server.
- **`floating_ip_address`** (*str*) – The IPv4 address of the Floating IP to assign. If specified, this Floating IP must already be allocated to the project.

`chi.server.`**`create_server`**(*server_name*, *reservation_id=None*, *key_name=None*, *network_id=None*, *network_name='sharednet1'*, *nics=[]*, *image_id=None*, *image_name='CC-CentOS7'*, *flavor_id=None*, *flavor_name=None*, *count=1*, *hypervisor_hostname=None*) → Union[NovaServer, list[NovaServer]]

Launch a new server instance.

Parameters

- **`server_name`** (*str*) – A name to give the server.
- **`reservation_id`** (*str*) – The ID of the Blazar reservation that will be used to select a target host node. It is required to make a reservation for bare metal server instances.
- **`key_name`** (*str*) – A key pair name to associate with the server. Any user holding the private key for the key pair will be able to SSH to the instance as the `cc` user. Defaults to the key specified by the "key_name" context variable.
- **`network_id`** (*str*) – The network ID to connect the server to. The server will obtain an IP address on this network when it boots.
- **`network_name`** (*str*) – The name of the network to connect the server to. If `network_id` is also set, that takes priority.
- **`nics`** (*list[dict]*) – ...
- **`image_id`** (*str*) – The image ID to use for the server's disk image.
- **`image_name`** (*str*) – The name of the image to user for the server's disk image. If `image_id` is also set, that takes priority. (Default `DEFAULT_IMAGE`.)
- **`flavor_id`** (*str*) – The flavor ID to use when launching the server. If not set, and no `flavor_name` is set, the first flavor found is used.
- **`flavor_name`** (*str*) – The name of the flavor to use when launching the server. If `flavor_id` is also set, that takes priority. If not set, and no `flavor_id` is set, the first flavor found is used.

- **count** (*int*) – The number of instances to launch. When launching bare metal server instances, this number must be less than or equal to the total number of hosts reserved. (Default 1).

Returns

The created server instance. If count was larger than 1, then a list of all created instances will be returned instead.

Raises ValueError – if an invalid count is provided.

chi.server.**delete_server**(*server_id*)
Delete a server by its ID.

Parameters **server_id** (*str*) – The ID of the server to delete.

chi.server.**detach_floating_ip**(*server_id*, *floating_ip_address*)
Remove an allocated Floating IP from a server by name.

Parameters

- **server_id** (*str*) – The name of the server.
- **floating_ip_address** (*str*) – The IPv4 address of the Floating IP to remove from the server.

chi.server.**get_flavor**(*ref*) → novaclient.v2.flavor_access.FlavorAccess
Get a flavor by its ID or name.

Parameters **ref** (*str*) – The ID or name of the flavor.

Returns The flavor matching the ID or name.

Raises **NotFound** – If the flavor could not be found.

chi.server.**get_flavor_id**(*name*) → str
Look up a flavor's ID from its name.

Parameters **name** (*str*) – The name of the flavor.

Returns The ID of the found flavor.

Raises **NotFound** – If the flavor could not be found.

chi.server.**get_server**(*ref*) → novaclient.v2.servers.Server
Get a server by its ID.

Parameters **ref** (*str*) – The ID or name of the server.

Returns The server matching the ID.

Raises **NotFound** – If the server could not be found.

chi.server.**get_server_id**(*name*) → str
Look up a server's ID from its name.

Parameters **name** (*str*) – The name of the server.

Returns The ID of the found server.

Raises **NotFound** – If the server could not be found.

chi.server.**list_flavors**() → list[NovaFlavor]
Get a list of all available flavors.

Returns A list of all flavors.

`chi.server.`**`list_servers`**`(`*`**kwargs`*`)` → list[NovaServer]

List all servers under the current project.

Parameters **`kwargs`** – Keyword arguments, which will be passed to `novaclient.v2.servers.list()`. For example, to filter by instance name, provide `search_opts={'name': 'my-instance'}`

Returns All servers associated with the current project.

`chi.server.`**`show_flavor`**`(`*`flavor_id`*`)` → novaclient.v2.flavor_access.FlavorAccess

Get a flavor by its ID.

Parameters **`flavor_id`** (*`str`*) – the ID of the flavor

Returns The flavor with the given ID.

`chi.server.`**`show_flavor_by_name`**`(`*`name`*`)` → novaclient.v2.flavor_access.FlavorAccess

Get a flavor by its name.

Parameters **`name`** (*`str`*) – The name of the flavor.

Returns The flavor with the given name.

Raises **`NotFound`** – If the flavor could not be found.

`chi.server.`**`show_server`**`(`*`server_id`*`)` → novaclient.v2.servers.Server

Get a server by its ID.

Parameters **`server_id`** (*`str`*) – the ID of the server

Returns The server with the given ID.

`chi.server.`**`show_server_by_name`**`(`*`name`*`)` → novaclient.v2.servers.Server

Get a server by its name.

Parameters **`name`** (*`str`*) – The name of the server.

Returns The server with the given name.

Raises **`NotFound`** – If the server could not be found.

`chi.server.`**`update_keypair`**`(`*`key_name=None`*`, `*`public_key=None`*`)` → novaclient.v2.keypairs.Keypair

Update a key pair's public key.

Due to how OpenStack Nova works, this requires deleting and re-creating the key even for public key updates. The key will not be re-created if it already exists and the fingerprints match.

Parameters

- **`key_name`** (*`str`*) – The name of the key pair to update. Defaults to value of the "key_name" context variable.
- **`public_key`** (*`str`*) – The public key to update the key pair to reference. Defaults to the contents of the file specified by the "keypair_public_key" context variable.

Returns The updated (or created) key pair.

`chi.server.`**`wait_for_active`**`(`*`server_id`*`, `*`timeout=1200`*`)`

Wait for the server to go in to the ACTIVE state.

If the server goes in to an ERROR state, this function will terminate. This is a blocking function.

Note: For bare metal servers, when the server transitions to ACTIVE state, this actually indicates it has started its final boot. It may still take some time for the boot to complete and interfaces e.g., SSH to come up.

If you want to wait for a TCP service like SSH, refer to `wait_for_tcp()`.

Parameters

- **server_id** (*str*) – The ID of the server.
- **timeout** (*int*) – The number of seconds to wait for before giving up. Defaults to 20 minutes.

`chi.server.wait_for_tcp(host, port, timeout=1200)`

Wait until a port on a server starts accepting TCP connections.

The implementation is taken from wait_for_tcp_port.py.

Parameters

- **host** (*str*) – The host that should be accepting connections. This can be either a Floating IP or a hostname.
- **port** (*int*) – Port number.
- **timeout** (*int*) – How long to wait before raising errors, in seconds. Defaults to 20 minutes.

Raises `TimeoutError` – If the port isn't accepting connection after time specified in *timeout*.

3.3.2 Object-oriented interface

The `Server` abstraction has been available historically for those who wish to use something with more of an OOP flavor.

`class chi.server.Server(id=None, lease=None, key=None, image='CC-CentOS7', **kwargs)`

A wrapper object referring to a server instance.

This class is helpful if you want to use a more object-oriented programming approach when building your infrastrucutre. With the Server abstraction, you can for example do the following:

```
with Server(lease=my_lease, image=my_image) as server:
    # When entering this block, the server is guaranteed to be
    # in the "ACTIVE" state if it launched successfully.
    server.associate_floating_ip()
    # Interact with the server (via, e.g., SSH), then...
# When the block exits, the server will be terminated and deleted
```

The above example uses a context manager. The class can also be used without a context manager:

```
# Triggers the launch of the server instance
server = Server(lease=my_lease, image=my_image)
# Wait for server to be active
server.wait()
server.associate_floating_ip()
# Interact with the server, then...
server.delete()
```

id

The ID of an existing server instance. Use this if you have already launched the instance and just want a convenient wrapper object for it.

Type str

`lease`
The Lease the instance will be launched under.

Type *Lease*

`key`
The name of the key pair to associate with the image. This is only applicable if launching the image; key pairs cannot be added to a server that has already been launched and wrapped via the `id` attribute.

Type str

`image`
The name or ID of the disk iage to use.

Type str

`name`
A name to give the new instance. (Defaults to an auto-generated name.)

Type str

`net_ids`
A list of network IDs to associate the instance with. The instance will obtain an IP address on each network during boot.

Note: For bare metal instances, the number of network IDs cannot exceed the number of enabled NICs on the bare metal node.

Type list[str]

`kwargs`
Additional keyword arguments to pass to Nova's server `create()` function.

`associate_floating_ip()`
Attach a floating IP to this server instance.

`delete()`
Delete this server instance.

`disassociate_floating_ip()`
Detach the floating IP attached to this server instance, if any.

`property error: bool`
Check if the instance is in an error state.

`property ready: bool`
Check if the instance is marked as active.

`rebuild(`*image_ref*`)`
Rebuild this server instance.

Note: For bare metal instances, this effectively redeploys to the host and overwrites the local disk.

`refresh()`
Poll the latest state of the server instance.

`property status: str`
Get the instance status.

wait()

Wait for the server instance to finish launching.

If the server goes into an error state, this function will return early.

3.4 chi.network

The *chi.network* module exposes a functional interface for interacting with the various networking capabilities of the testbed.

chi.network.**add_port_to_router**(*router_id*, *port_id*)

Add a port to a router.

Parameters

- **router_id** (*str*) – The router ID.
- **port_id** (*str*) – The port ID.

chi.network.**add_port_to_router_by_name**(*router_name*, *port_name*)

Add a port to a router, referencing the router and port by name.

Parameters

- **router_name** (*str*) – The router name.
- **port_name** (*str*) – The port name.

chi.network.**add_route_to_router**(*router_id*, *cidr*, *nexthop*)

Add a new route to a router.

Parameters

- **router_id** (*str*) – The router ID.
- **cidr** (*str*) – The destination subnet CIDR for the route.
- **nexthop** (*str*) – The nexthop address for the route.

chi.network.**add_routes_to_router**(*router_id*, *routes*)

Add a set of routes to a router.

Parameters

- **router_id** (*str*) – The router ID.
- **routes** (*list[dict]*) – A list of routes to add. The list is expected to consist of items with a 'destination' and 'nexthop' key, e.g.:

```
[
   {'destination': '10.0.0.0/24', 'nexthop': '10.0.0.1'},
   {'destination': '10.0.1.0/24', 'nexthop': '10.0.1.1'}
]
```

chi.network.**add_subnet_to_router**(*router_id*, *subnet_id*)

Add a subnet to a router.

Parameters

- **router_id** (*str*) – The router ID.
- **subnet_id** (*str*) – The subnet ID.

`chi.network.`**`add_subnet_to_router_by_name`**(*router_name*, *subnet_name*)

Add a subnet to a router, referencing the router and subnet by name.

Parameters

- **`router_name`** (*`str`*) – The router name.
- **`subnet_name`** (*`str`*) – The subnet name.

`chi.network.`**`bind_floating_ip`**(*ip_address*, *port_id=None*, *fixed_ip_address=None*)

Directly assign a Floating IP to an existing port/address.

Note: If you just want to attach a Floating IP to a server instance, the *`chi.server.associate_floating_ip()`* function is simpler.

Parameters

- **`ip_address`** (*`str`*) – The Floating IP address.
- **`port_id`** (*`str`*) – The ID of the port to bind to.
- **`fixed_ip_address`** (*`str`*) – The address in the port to bind to. This is only required if the port has multiple IP addresses assigned; by default the first IP in a port is bound.

`chi.network.`**`create_network`**(*network_name*, *of_controller_ip=None*, *of_controller_port=None*, *vswitch_name=None*, *provider='physnet1'*, *port_security_enabled=True*) → dict

Create a network.

For an OpenFlow network include the IP and port of an OpenFlow controller on Chameleon or accessible through the public Internet. Include a virtual switch name if you plan to add additional private VLANs to this switch. Additional VLANs can be connected using a dedicated port corresponding to the VLAN tag and can be conrolled using a valid OpenFlow controller.

Parameters

- **`network_name`** (*`str`*) – The new network name.
- **`of_controller_ip`** (*`str`*) – the IP of the optional OpenFlow controller. The IP must be accessible on the public Internet.
- **`of_controller_port`** (*`str`*) – the port of the optional OpenFlow controller.
- **`vswitch_name`** (*`str`*) – The virtual switch to use name.
- **`provider`** (*`str`*) – the provider network to use when specifying stitchable VLANs (i.e. ExoGENI). Default: 'physnet1'

`chi.network.`**`create_port`**(*port_name*, *network_id*, *fixed_ips=None*, *subnet_id=None*, *ip_address=None*, *port_security_enabled=True*) → dict

Create a new port on a network.

This function has a short-form and a long-form invocation. In the short form, you can specify `subnet_id` and `ip_address` to give the port a single assignment on a subnet. In the long form you can specify `fixed_ips` to define multiple assignments.

Parameters

- **`port_name`** (*`str`*) – The name to give the new port.
- **`network_id`** (*`str`*) – The ID of the network that the port will be connected to.

- **fixed_ips** (*list[dict]*) – A list of IP assignments to give to the port on various subnets. Each assignment must at minimum have a `subnet_id` defined. An optional `ip_address` can be included on an assignment to specify the exact IP address to assign. Otherwise, one is chosen automatically from the available IPs on the subnet. There can be multiple assignments (i.e., IPs) on a single subnet.
- **subnet_id** (*str*) – The ID of the subnet that the port will be allocated on. The port will be automatically assigned an IP address on this subnet, unless the `ip_address` parameter is provided.

  Note: This parameter is ignored if `fixed_ips` is set.

- **ip_address** (*str*) – The IP address to assign the port, if a specific IP address is desired. By default an IP address is automatically picked from the target subnet.

  Note: This parameter is ignored if `fixed_ips` is set.

- **port_security_enabled** (*bool*) – Whether to enable port security. In general this should be kept on. (Default True).

Returns The created port representation.

`chi.network.`**`create_router`**(*router_name*, *gw_network_name=None*) → dict

Create a router, with or without a public gateway.

Parameters

- **router_name** (*str*) – The new router name.
- **gw_network_name** (*str*) – The name of the public gateway requested to provide subnets connected this router NAT to the Internet.

Returns The created router representation.

`chi.network.`**`create_subnet`**(*subnet_name*, *network_id*, *cidr='192.168.1.0/24'*, *allocation_pool_start=None*, *allocation_pool_end=None*, *gateway_ip=None*) → dict

Create a subnet on a network.

Parameters

- **subnet_name** (*str*) – The name to give the new subnet.
- **network_id** (*str*) – The network to associate the subnet with ID.
- **cidr** (*str*) – The subnet's IPv4 CIDR range. (Default 192.168.1.0/24)
- **gateway_ip** (*str*) – The subnet's gateway address. If not defined, the first address in the subnet will be automatically chosen as the gateway.

Returns The new subnet representation.

`chi.network.`**`delete_network`**(*network_id*)

Delete the network.

Note: This does not perform a full teardown of the network, including removing subnets and ports. It will only succeed if the network does not have any attached entities. See `nuke_network()` for a more complete teardown function.

Parameters **`network_id`** (*`str`*) – The network ID.

`chi.network.`**`delete_port`**`(`*port_id*`)`
Delete the port.

Parameters **`port_id`** (*`str`*) – The port ID.

`chi.network.`**`delete_router`**`(`*router_id*`)`
Delete the router.

Parameters **`router_id`** (*`str`*) – The router ID.

`chi.network.`**`delete_subnet`**`(`*subnet_id*`)`
Delete the subnet.

Parameters **`subnet_id`** (*`str`*) – The subnet ID.

`chi.network.`**`get_floating_ip`**`(`*ip_address*`)` → dict
Get the floating IP representation for an IP address.

Parameters **`ip_address`** (*`str`*) – The IP address of the floating IP.

Returns The floating IP representation.

`chi.network.`**`get_free_floating_ip`**`(`*allocate=True*`)` → dict
Get the first unallocated floating IP available to your project.

Parameters **`allocate`** (*`bool`*) – Whether to allocate a new floating IP if there are no Floating IPs currently free in your project. Defaults to True.

Returns The free floating IP representation.

`chi.network.`**`get_network`**`(`*ref*`)` → dict
Get a network by its name or ID.

Parameters **`ref`** (*`str`*) – The name or ID of the network.

Returns The network representation.

Raises **`RuntimeError`** – If the network could not be found, or multiple networks were returned for the search term.

`chi.network.`**`get_network_id`**`(`*name*`)` → str
Look up a network's ID from its name.

Parameters **`name`** (*`str`*) – The network name.

Returns The network's ID, if found.

Raises **`RuntimeError`** – If the network could not be found, or multiple networks were returned for the search term.

`chi.network.`**`get_port`**`(`*ref*`)` → dict
Get a port by its name or ID.

Parameters **`ref`** (*`str`*) – The name or ID of the port.

Returns The port representation.

Raises **`RuntimeError`** – If the port could not be found, or multiple ports were returned for the search term.

`chi.network.`**`get_port_id`**`(`*name*`)` → str
Look up a port's ID from its name.

Parameters **`name`** (*`str`*) – The port name.

Returns The port's ID, if found.

Raises `RuntimeError` – If the port could not be found, or multiple ports were returned for the search term.

`chi.network.`**`get_router`**`(`*ref*`)` → dict

Get a router by its name or ID.

Parameters **`ref`** (*`str`*) – The name or ID of the router.

Returns The router representation.

Raises `RuntimeError` – If the router could not be found, or multiple routers were returned for the search term.

`chi.network.`**`get_router_id`**`(`*name*`)` → str

Look up a router's ID from its name.

Parameters **`name`** (*`str`*) – The router name.

Returns The router's ID, if found.

Raises `RuntimeError` – If the router could not be found, or multiple routers were returned for the search term.

`chi.network.`**`get_subnet`**`(`*ref*`)` → dict

Get a subnet by its name or ID.

Parameters **`ref`** (*`str`*) – The name or ID of the subnet.

Returns The subnet representation.

Raises `RuntimeError` – If the subnet could not be found, or multiple subnets were returned for the search term.

`chi.network.`**`get_subnet_id`**`(`*name*`)` → str

Look up a subnet's ID from its name.

Parameters **`name`** (*`str`*) – The subnet name.

Returns The subnet's ID, if found.

Raises `RuntimeError` – If the subnet could not be found, or multiple subnets were returned for the search term.

`chi.network.`**`list_floating_ips`**`()` → list[dict]

List all floating ips associated with the current project.

Returns A list of all the found floating ips.

`chi.network.`**`list_networks`**`()` → list[dict]

List all networks associated with the current project.

Returns A list of all the found networks.

`chi.network.`**`list_ports`**`()` → list[dict]

List all ports associated with the current project.

Returns A list of all the found ports.

`chi.network.`**`list_routers`**`()` → list[dict]

List all routers associated with the current project.

Returns A list of all the found routers.

`chi.network.`**`list_subnets`**`()` → list[dict]

List all subnets associated with the current project.

Returns A list of all the found subnets.

`chi.network.`**`nuke_network`**`(`*network_ref: str*`)`

Completely tear down the network.

Cleanly tearing down an OpenStack network representation involves a few separate steps:

1. Detach the network's subnets from the router.
2. Delete the router.
3. Delete the subnet(s).
4. Delete the network.

This function performs all of those steps for you.

Note: This function will not work well for very advance networks, perhaps those connected to multiple routers. You should perform your own cleanup if your network's subnets are attached to multiple routers.

Parameters **`network_ref`** (*`str`*) – The network name or ID.

`chi.network.`**`remove_all_routes_from_router`**`(`*router_id*`)`

Remove all routes from the router.

Parameters **`router_id`** (*`str`*) – The router ID.

`chi.network.`**`remove_port_from_router`**`(`*router_id*, *port_id*`)`

Remove a port from the router.

Parameters

- **`router_id`** (*`str`*) – The router ID.
- **`port_id`** (*`str`*) – The port ID.

`chi.network.`**`remove_route_from_router`**`(`*router_id*, *cidr*, *nexthop*`)`

Remove a single route from the router.

Parameters

- **`router_id`** (*`str`*) – The router ID.
- **`cidr`** (*`str`*) – The destination subnet CIDR for the route.
- **`nexthop`** (*`str`*) – The nexthop address for the route.

`chi.network.`**`remove_routes_from_router`**`(`*router_id*, *routes*`)`

Remove a set of routes from a router.

Parameters

- **`router_id`** (*`str`*) – The router ID.
- **`routes`** (*`list[dict]`*) – A list of routes to remove. The list is expected to consist of items with a 'destination' and 'nexthop' key, e.g.:

```
[
   {'destination': '10.0.0.0/24', 'nexthop': '10.0.0.1'},
   {'destination': '10.0.1.0/24', 'nexthop': '10.0.1.1'}
]
```

`chi.network.`**`remove_subnet_from_router`**`(`*router_id*, *subnet_id*`)`

Remove a subnet from the router.

Parameters

- **`router_id`** (*`str`*) – The router ID.
- **`subnet_id`** (*`str`*) – The subnet ID.

3.4.1 Wizards

There are additionally some functions that tie together several common tasks.

`class` `chi.network.`**`wizard`**

A collection of "wizard" functions.

These utility functions are very opinionated but can reduce boilerplate.

`static` **`create_network`**`(`*name_prefix*, *of_controller_ip=None*, *of_controller_port=None*, *gateway=False*`)`

Create a network and subnet, and connect the subnet to a new router.

Parameters

- **`name_prefix`** (*`str`*) – The common name prefix for all created entities.
- **`of_controller_ip`** (*`str`*) – The OpenFlow controller IP, if using.
- **`of_controller_port`** (*`int`*) – The OpenFlow controller port, if using.
- **`gateway`** (*`bool`*) – Whether to add a WAN gateway to the router. Routers with a WAN gateway are able to NAT to the Internet.

Returns The created network representation.

`static` **`delete_network`**`(`*name_prefix*`)`

Delete a network created via :func:`wizard.create_network`.

Parameters **`name_prefix`** (*`str`*) – The common name prefix for all created entities.

3.5 chi.image

The *`chi.image`* module exposes a functional interface for interacting with disk images.

`chi.image.`**`get_image`**`(`*ref*`)`

Get an image by its ID or name.

Parameters **`ref`** (*`str`*) – The ID or name of the image.

Returns The image matching the ID or name.

Raises **`NotFound`** – If the image could not be found.

`chi.image.`**`get_image_id`**`(`*name*`)`

Look up an image's ID from its name.

Parameters **`name`** (*`str`*) – The name of the image.

Returns The ID of the found image.

Raises **`ValueError`** – If the image could not be found, or if multiple images matched the name.

`chi.image.`**`list_images`**`()`

List all images under the current project.

Returns All images associated with the current project.

3.6 chi.container

The `chi.container` module exposes a functional interface for interacting with application containers.

Important: Currently, only the CHI@Edge site support container operations.

`chi.container.`**`create_container`**`(`*name: str, image: str = None, image_driver: str = 'docker', device_profiles: list[str] = None, environment: dict = None, exposed_ports: list[str] = [], runtime: str = None, nets: list[dict] = None, network_id: str = None, network_name: str = 'containernet1', reservation_id: str = None, start: bool = True, start_timeout: int = None, **kwargs*`)` → Container

Create a container instance.

Parameters

- **`name`** (*`str`*) – The name to give the container.
- **`image`** (*`str`*) – The Docker image, with or without tag information. If no tag is provided, "latest" is assumed.
- **`image_driver`** (*`str`*) – The image storage driver to use to retrieve the image. Defaults to "docker", meaning the image is assumed to be a Docker registry repository. Specify "glance" to launch a snapshot image by passing the Glance Image ID in the `image` argument.
- **`device_profiles`** (*`list[str]`*) – An optional list of device profiles to request be configured on the container when it is created. Edge devices may have differing sets of supported device profiles, so it is important to understand which profiles are supported by the target device for your container.
- **`environment`** (*`dict`*) – A set of environment variables to pass to the container.
- **`exposed_ports`** (*`list[str]`*) – A list of ports to expose on the container. TCP or UDP can be provided with a slash prefix, e.g., "80/tcp" vs. "53/udp". If no protocol is provided, TCP is assumed.
- **`nets`** (*`list[dict]`*) – A set of network configurations. This is an advanced invocation; typically `network_id` or `network_name` should be enough, and is much simpler. Refer to the Zun documentation for information about this parameter.
- **`network_id`** (*`str`*) – The ID of a network to launch the container on.
- **`network_name`** (*`str`*) – The name of a network to launch the container on. This has no effect if `network_id` is already provided. Default "containernet1".
- **`host`** (*`str`*) – The Zun host to launch a container on. If not specified, the host is chosen by Zun.
- **`runtime`** (*`str`*) – The container runtime to use. This should only be overridden when explicitly launching containers onto a host/platform requiring a separate runtime to, e.g., pass-through GPU devices, such as the "nvidia" runtime provided by NVIDIA Jetson Nano/TX2.
- **`start`** (*`bool`*) – Whether to automatically start the container after it is created. Default True.
- **`**kwargs`** – Additional keyword arguments to send to the Zun client's container create call.

`chi.container.`**`destroy_container`**(*container_ref: str*)

Delete the container.

This will automatically stop the container if it is currently running.

Parameters **`container_ref`** (*`str`*) – The name or ID of the container.

`chi.container.`**`execute`**(*container_ref: str*, *command: str*) → dict

Execute a one-off process inside a running container.

Parameters

- **`container_ref`** (*`str`*) – The name or ID of the container.
- **`command`** (*`str`*) – The command to run.

Returns A summary of the output of the command, with "output" and "exit_code".

`chi.container.`**`get_container`**(*container_ref: str*) → Container

Get a container's information.

Parameters

- **`container_ref`** (*`str`*) – The name or ID of the container.
- **`tag`** (*`str`*) – An optional version to tag the container image with. If not defined, defaults to "latest".

Returns The container, if found.

`chi.container.`**`get_logs`**(*container_ref: str*, *stdout=True*, *stderr=True*)

Print all logs outputted by the container.

Parameters

- **`container_ref`** (*`str`*) – The name or ID of the container.
- **`stdout`** (*`bool`*) – Whether to include stdout logs. Default True.
- **`stderr`** (*`bool`*) – Whether to include stderr logs. Default True.

Returns

A string containing all log output. Log lines will be delimited by newline characters.

`chi.container.`**`list_containers`**() → list[Container]

List all containers owned by this project.

Returns A list of containers.

`chi.container.`**`snapshot_container`**(*container_ref: str*, *repository: str*, *tag: str = 'latest'*) → str

Create a snapshot of a running container.

This will store the container's file system in Glance as a new Image. You can then specify the Image ID in container create requests.

Parameters

- **`container_ref`** (*`str`*) – The name or ID of the container.
- **`repository`** (*`str`*) – The name to give the snapshot.
- **`tag`** (*`str`*) – An optional version tag to give the snapshot. Defaults to "latest".

`chi.container.`**`upload`**(*container_ref: str*, *source: str*, *dest: str*) → dict

Upload a file or directory to a running container.

Parameters

- **container_ref** (*str*) – The name or ID of the container.
- **source** (*str*) – The (local) path to the file or directory to upload.
- **dest** (*str*) – The (container) path to upload the file or directory to.

`chi.container.wait_for_active`(*container_ref: str*, *timeout: int = 120*) → Container

Wait for a container to transition to the running state.

Parameters

- **container_ref** (*str*) – The name or ID of the container.
- **timeout** (*int*) – How long to wait before issuing a TimeoutError.

Raises `TimeoutError` – if the timeout was reached before the container started.

Returns The container representation.

3.7 Launching a bare metal instance

First, select which project and site you wish to authenticate against.

```
[ ]: import chi

     chi.use_site('CHI@UC')
     # Set to your project's charge code
     chi.set('project_name', 'CH-XXXXXX')
```

3.7.1 Launch a bare metal instance.

Functions used in this example:

- create_server
- get_node_reservation

```
[ ]: from chi.lease import get_node_reservation
     from chi.server import create_server

     # We assume a lease has already been created, for example with
     # ``chi.lease.create_lease```
     lease_name = "my_lease"
     server_name = "my_server"
     reservation_id = get_node_reservation(lease_name)
     server = create_server(server_name, reservation_id=reservation_id)
```

3.7.2 Wait for a server's port to come up before proceeding.

Sometimes you want to interact with the server over a remote interface and need to wait until it's up and accepting connections. The :func:~`chi.server.wait_for_tcp` function allows you to do just that. This example also illustrates how you can bind a Floating IP (public IP) to the server so it can be reached over the internet.

Functions used in this example:

- associate_floating_ip
- wait_for_tcp

```
[ ]: from chi.server import associate_floating_ip, wait_for_tcp

     # Note: this is a placeholder server ID. Yours will be different!
     # server_id can be obtained like `server.id` if you created the server
     # with `create_server`. It can also be obtained via `get_server_id(name)`
     server_id = "6b2bae1e-0311-493f-836c-a9da0cb9e0c0"
     ip = associate_floating_ip(server_id)

     # Wait for SSH connectivity over port 22
     wait_for_tcp(ip, port=22)
```

3.8 Launching a container

First, select which project and site you wish to authenticate against.

```
[ ]: import chi

     chi.use_site('CHI@UC')
     # Set to your project's charge code
     chi.set('project_name', 'CH-XXXXXX')
```

3.8.1 Launch a container.

Functions used in this example:

- create_container
- get_device_reservation

```
[ ]: from chi.lease import get_device_reservation
     from chi.container import create_container

     # We assume a lease has already been created, for example with
     # ``chi.lease.create_lease``
```

(continues on next page)

(continued from previous page)

```
lease_name = "my_lease"
container_name = "my_container"
reservation_id = get_device_reservation(lease_name)
container = create_container(
    container_name,
    image="centos:8",
    reservation_id=reservation_id,
)
```

3.9 Making a reservation

First, select which project and site you wish to authenticate against.

```
[ ]: import chi

chi.use_site('CHI@UC')
# Set to your project's charge code
chi.set('project_name', 'CH-XXXXXX')
```

3.9.1 Reserve a bare metal node.

Multiple nodes can be reserved at once by changing the `count` variable. This example makes a reservation for the "compute_skylake" node type. See here for a complete list of node types available currently.

Functions used in this example:

- add_node_reservation
- lease_duration
- create_lease

```
[ ]: from chi.lease import lease_duration, add_node_reservation, create_lease

lease_name = "myLease"
node_type = "compute_skylake"
start_date, end_date = lease_duration(days=1)

# Build list of reservations (in this case there is only one reservation)
reservations = []
add_node_reservation(reservations, count=1, node_type=node_type)
# Create the lease
lease = create_lease(lease_name, reservations, start_date=start_date,
                     end_date=end_date)
```

3.9.2 Reserve a floating IP.

While it's possible to allocate a floating IP ad hoc from Chameleon most of the time, there are a limited amount of IPs and they are sometimes exhausted. You can reserve a floating IP to ensure you have access to one to attach to your experimental nodes to allow, e.g., external SSH connectivity.

See here for some tips on how to make the most out of a single floating IP, which can help you avoid excessive charges.

Functions used in this example:

- add_fip_reservation
- lease_duration
- create_lease

[ ]:
```python
from chi.lease import lease_duration, add_fip_reservation, create_lease

lease_name = "myLease"
start_date, end_date = lease_duration(days=1)

# Build list of reservations (in this case there is only one reservation)
reservation_list = []
add_fip_reservation(reservation_list, count=1)

# Create the lease
lease = create_lease(lease_name, reservation_list, start_date=start_date,
                     end_date=end_date)
```

3.9.3 Reserve a VLAN segment.

This example illustrates how to reserve an isolated VLAN in order to ensure your network experiment is not subject to cross-traffic from other experimenters.

This is also how you reserve stitchable VLANs provided through ExoGENI. For these VLANs, you must set `physical_network` to "exogeni".

Functions used in this example:

- add_network_reservation
- lease_duration
- create_lease

[ ]:
```python
from chi.lease import lease_duration, add_network_reservation, create_lease

lease_name = "myLease"
network_name = f"{lease_name}Network"
of_controller_ip = None
of_controller_port = None
vswitch_name = None
```

(continues on next page)

(continued from previous page)

```
physical_network = "physnet1"
start_date, end_date = lease_duration(days=1)

# Build list of reservations (in this case there is only one reservation)
reservations = []
add_network_reservation(reservations,
                        network_name=network_name,
                        of_controller_ip=of_controller_ip,
                        of_controller_port=of_controller_port,
                        vswitch_name=vswitch_name,
                        physical_network=physical_network)

# Create the lease
lease = create_lease(lease_name, reservations, start_date=start_date,
                     end_date=end_date)
```

3.9.4 Reserve multiple types of resources in a single lease.

Functions used in this example:

- add_node_reservation
- add_network_reservation
- add_fip_reservation
- lease_duration
- create_lease

[ ]:
```
from chi.lease import (
    lease_duration, add_node_reservation, add_network_reservation,
    add_fip_reservation, create_lease)

lease_name = "myLease"
start_date, end_date = lease_duration(days=1)

# Build list of reservations
reservations = []
add_node_reservation(reservations, count=1, node_type="compute_skylake")
add_network_reservation(reservations, network_name=f"{lease_name}Network")
add_fip_reservation(reservations, count=1)

# Create the lease
lease = create_lease(lease_name, reservations, start_date=start_date,
                     end_date=end_date)
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX