php-opencloud Documentation

Release 1.12.1

Jamie Hannaford

May 25, 2017

Contents

CHAPTER 1

Requirements

- PHP 7
- cURL extension

CHAPTER 2

Installation

You must install this library through Composer:

```
composer require php-opencloud/openstack
```

If you do not have Composer installed, please read the Composer installation instructions.

Once you have installed the SDK as a dependency of your project, you will need to load Composer's autoloader (which registers all the required namespaces). To do this, place the following line of PHP code at the top of your application's PHP files:

```
require 'vendor/autoload.php';
```

This assumes your application's PHP files are located in the same folder as `vendor/`. If your files are located elsewhere, please supply the path to vendor/autoload.php in the require statement above.

CHAPTER 3

Supported services

Block Storage v2

Volumes

List volumes

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volumes = $service->listVolumes();

foreach ($volumes as $volume) {
    /** @var $volume \OpenStack\BlockStorage\v2\Models\Volume */
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Each iteration will return a php:class:*Volume* instance <OpenStack/BlockStorage/v2/Models/Volume.html>.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with

generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Detailed information

By default, only the `id`, `links` and `name` attributes are returned. To return *all* information for a flavor, you must enable detailed information, like so:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

foreach ($service->listVolumes(true) as $volume) {
    /** @var $volume \OpenStack\BlockStorage\v2\Models\Volume */
}
```

Create volume

The only attributes that are required when creating a volume are a size in GiB. The simplest example would therefore be this:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->createVolume([
    'description'      => '{description}',
    'size'             => '{size}',
    'name'             => '{name}',
    'volumeType'       => '{volumeType}',
    'metadata'         => ['{key1}' => '{val1}'],
]);
```

You can further configure your new volume, however, by following the below sections, which instruct you how to add specific functionality.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Create from image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->createVolume([
    'description'    => '{description}',
    'size'           => '{size}',
    'name'           => '{name}',
    'sourceVolumeId' => '{snapshotId}',
]);
```

Create from snapshot

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->createVolume([
    'description' => '{description}',
    'size'        => '{size}',
    'name'        => '{name}',
    'snapshotId'  => '{snapshotId}',
]);
```

Create from source volume

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->createVolume([
    'description' => '{description}',
    'size'        => '{size}',
    'name'        => '{name}',
    'imageId'     => '{snapshotId}',
]);
```

Retrieve volume details

When retrieving a volume, sometimes you only want to operate on it - say to update or delete it. If this is the case, then there is no need to perform an initial GET request to the API:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->getVolume('{volumeId}');
```

If, however, you *do* want to retrieve all the details of a remote volume from the API, you just call:

```
$volume->retrieve();
```

which will update the state of the local object. This gives you an element of control over your app's performance.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update volume

The first step when updating a volume is modifying the attributes you want updated. By default, only a volume's name and description can be edited.

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->getVolume('{volumeId}');

$volume->name        = '{newName}';
$volume->description = '{newDescription}';

$volume->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete volume

To permanently delete a volume:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volume = $service->getVolume('{volumeId}');
$volume->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Volume Types

Listing volume types

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
```

```
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volumeTypes = $service->listVolumeTypes();

foreach ($volumeTypes as $volumeType) {
    /** @var $volumeType \OpenStack\BlockStorage\v2\VolumeType */
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Each iteration will return a `VolumeType` instance <OpenStack/BlockStorage/v2/Models/VolumeType.html>.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Create volume type

The only attributes that are required when creating a volume are a name. The simplest example would therefore be this:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volumeType = $service->createVolumeType([
    'name' => '{name}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Retrieve details of a volume type

When retrieving a volume type, sometimes you only want to operate on it - say to update or delete it. If this is the case, then there is no need to perform an initial GET request to the API:

```
<?php
```

```
require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volumeType = $service->getVolumeType('{volumeTypeId}');
```

If, however, you *do* want to retrieve all the details of a remote volume type from the API, you just call:

```
$volumeType->retrieve();
```

which will update the state of the local object. This gives you an element of control over your app's performance.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update a volume type

The first step when updating a volume type is modifying the attributes you want updated. By default, only a volume type's name can be edited.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volumeType = $service->getVolumeType('{volumeTypeId}');
$volumeType->name = '{newName}';
$volumeType->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete volume type

To permanently delete a volume type:

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->blockStorageV2();

$volumeType = $service->getVolumeType('{volumeTypeId}');
$volumeType->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Snapshots

Compute v2

Servers

Listing servers

To list a collection of servers, you run:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$servers = $compute->listServers(['imageId' => '{imageId}']);

foreach ($servers as $server) {
}
```

Each iteration will return a :apiref:Server instance <OpenStack/Compute/v2/Models/Server.html>.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Detailed information

By default, only the id, links and name attributes are returned by the server. To return *all* information for a server, you must enable detailed information, like so:

```
$servers = $service->listServers(true);
```

Filtering collections

By default, every server will be returned by the remote API. To filter the returned collection, you can provide query parameters which are documented in the reference documentation.

```
use OpenStack\Common\DateTime;

$servers = $service->listServers(false, [
    'changesSince' => DateTime::factory('yesterday')->toIso8601(),
    'flavorId'     => 'performance1-1',
]);
```

Create a server

The only attributes that are required when creating a server are a name, flavor ID and image ID. The simplest example would therefore be this:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$options = [
    // Required
    'name'     => '{serverName}',
    'imageId'  => '{imageId}',
    'flavorId' => '{flavorId}',

    // Required if multiple network is defined
    'networks'  => [
        ['uuid' => '{networkId}']
    ],
```

```
    // Optional
    'metadata' => ['foo' => 'bar'],
    'userData' => base64_encode('echo "Hello World. The time is now $(date -R)!" |␣
↪tee /root/output.txt')
];

// Create the server
/**@var OpenStack\Compute\v2\Models\Server $server */
$server = $compute->createServer($options);
```

You can further configure your new server, however, by following the below sections, which instruct you how to add specific functionality. They are interoperable and can work together.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Security groups

You can associate your new server with pre-existing security groups by specifying their _name_. One server can be associated with multiple security groups:

```
$options['securityGroups'] = ['secGroup1', 'default', 'secGroup2'];
```

Networks

By default, the server instance is provisioned with all isolated networks for the tenant. You can, however, configure access by specifying which networks your VM is connected to. To do this, you can either:

- specify the UUID of a Neutron network. This is required if you omit the port ID.
- specify the UUID of a Neutron port. This is required if you omit the network ID. The port must exist and be in a DOWN state.

```
// Specifying the network
$options['networks'] = [
    ['uuid' => '{network1Id}'],
    ['uuid' => '{network2Id}'],
];

// Or, specifying the port:
$options['networks'] = [
    ['port' => '{port1Id}'],
    ['port' => '{port2Id}'],
];
```

External devices and boot from volume

This option allows for the booting of the server from a volume. If specified, the volume status must be available, and the volume attach_status in the OpenStack Block Storage DB must be detached.

For example, to boot a server from a Cinder volume:

```
$options['blockDeviceMapping'] = [
    [
        'deviceName'      => '/dev/sda1',
        'sourceType'      => 'volume',
        'destinationType' => 'volume',
        'uuid'            => '{volumeId}',
        'bootIndex'       => 0,
    ]
];
```

Personality files

Servers, as they're created, can be injected with arbitrary file data. To do this, you must specify the path and file contents (text only) to inject into the server at launch. The maximum size of the file path data is 255 bytes. The maximum limit refers to the number of bytes in the decoded data and not the number of characters in the encoded data.

The contents *must* be base-64 encoded.

```
$options['personality'] = [
    'path'     => '/etc/banner.txt',
    'contents' => base64_encode('echo "Hi!";'),
];
```

Metadata

The API also supports the ability to label servers with arbitrary key/value pairs, known as metadata. To specify this when the server is launched, use this option:

```
$options['metadata'] = [
    'foo' => 'bar',
    'baz' => 'bar',
];
```

Retrieve a server

When retrieving a server, sometimes you only want to operate on it - say to update or delete it. If this is the case, then there is no need to perform an initial GET request to the server:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer(['id' => '{serverId}']);

// By default, this will return an empty Server object and NOT hit the API.
// This is convenient for when you want to use the object for operations
// that do not require an initial GET request. To retrieve the server's details,
// run the following, which *will* call the API with a GET request:

$server->retrieve();
```

If, however, you *do* want to retrieve all the details of a remote server from the API, you just call:

```
$server->retrieve();
```

which will update the state of the local object. This gives you an element of control over your app's performance.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update a server

The first step when updating a server is modifying the attributes you want updated. By default, only a server's name, IPv4 and IPv6 IPs, and its auto disk config attributes can be edited.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->name = '{newName}';
$server->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete a server

To permanently delete a server:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

/**@var OpenStack\Compute\v2\Models\Server $server */
$server = $compute->getServer(['id' => '{serverId}']);

$server->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Retrieve metadata

This operation will retrieve the existing metadata for a server:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer(['id' => '{serverId}']);

$metadata = $server->getMetadata();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Reset metadata

This operation will _replace_ all existing metadata with whatever is provided in the request. Any existing metadata not specified in the request will be deleted.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->resetMetadata([
    'key' => 'value',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Merge metadata

This operation will _merge_ specified metadata with what already exists. Existing values will be overriden, new values will be added. Any existing keys that are not specified in the request will remain unaffected.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer(['id' => '{serverId}']);

$server->mergeMetadata([
    'key' => 'value'
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Retrieve metadata item

This operation allows you to retrieve the value for a specific metadata item:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer(['id' => '{serverId}']);

$metadataItem = $server->getMetadataItem('{metadataItem}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete metadata item

This operation allows you to remove a specific metadata item:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer(['id' => '{serverId}']);

$server->deleteMetadataItem('key');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Change root password

This operation will replace the root password for a server.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->changePassword('{newPassword}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Reboot server

This operation will reboot a server. Please be aware that you must specify whether you want to initiate a HARD or SOFT reboot (you specify this as a string argument).

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->reboot();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Rebuild server

Rebuilding a server will re-initialize the booting procedure for the server and effectively reinstall the operating system. It will shutdown, re-image and then reboot your instance. Any data saved on your instance will be lost when the rebuild is performed.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->rebuild([
    'imageId'   => '{imageId}',
    'name'      => '{newName}',
    'adminPass' => '{adminPass}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Resize server

You can resize the flavor of a server by performing this operation. As soon the operation completes, the server will transition to a VERIFY_RESIZE state and a VM status of RESIZED. You will either need to confirm or revert the resize in order to continue.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
```

```
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->resize('{flavorId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Confirm server resize

Once a server has been resized, you can confirm the operation by calling this. The server must have the status of VERIFY_RESIZE and a VM status of RESIZED. Once this operation completes, the server should transition to an ACTIVE state and a migration status of confirmed.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->confirmResize();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Revert server resize

Once a server has been resized, you can revert the operation by calling this. The server must have the status of VERIFY_RESIZE and a VM status of RESIZED. Once this operation completes, the server should transition to an ACTIVE state and a migration status of reverted.

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer([
    'id' => '{serverId}',
]);

$server->revertResize();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Create server image

This operation will create a new server image. The only required option is the new image's name. You may also specify additional metadata:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$server = $compute->getServer(['id' => '{serverId}']);

$server->createImage([
    'name' => '{imageName}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List server IP addresses

To list all the addresses for a specified server or a specified server and network:

```
$ipAddresses = $server->listAddresses();

$public  = $ipAddresses['public'];
$private = $ipAddresses['private'];
```

You can also refine by network label:

```
$ipAddresses = $server->listAddresses([
    'networkLabel' => '{networkLabel}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Further links

- Reference docs for Server class
- API docs

Flavors

List flavors

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$flavors = $compute->listFlavors();

foreach ($flavors as $flavor) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Each iteration will return a `Flavor` instance <OpenStack/Compute/v2/Models/Flavor.html>.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Detailed information

By default, only the `id`, `links` and `name` attributes are returned. To return *all* information for a flavor, you must enable detailed information, like so:

```
$flavors = $service->listFlavors(true);
```

Retrieve a flavor

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$flavor = $compute->getFlavor(['id' => '{flavorId}']);

// By default, this will return an empty Flavor object and NOT hit the API.
// This is convenient for when you want to use the object for operations
// that do not require an initial GET request. To retrieve the flavor's details,
// run the following, which *will* call the API with a GET request:

$flavor->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

When retrieving a flavor, sometimes you only want to operate on it. If this is the case, then there is no need to perform an initial `GET` request to the server:

```
// Get an unpopulated object
$flavor = $service->getFlavor(['id' => '{flavorId}']);
```

If, however, you *do* want to retrieve all the details of a remote flavor from the API, you just call:

```
$flavor->retrieve();
```

which will update the state of the local object. This gives you an element of control over your app's performance.

Images

List images

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$images = $compute->listImages(['status' => 'ACTIVE']);

foreach ($images as $image) {
}
```

Each iteration will return an :apiref:Image instance <OpenStack/Compute/v2/Models/Image.html>_.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Detailed information

By default, only the id, links and name attributes are returned. To return *all* information for an image, you must enable detailed information, like so:

```
$images = $service->listImages(true);
```

Retrieve an image

When retrieving an image, sometimes you only want to operate on it - say to update or delete it. If this is the case, then there is no need to perform an initial GET request to the server:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
```

```
        'password' => '{password}'
    ],
    'scope'    => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$image = $compute->getImage(['id' => '{imageId}']);

// By default, this will return an empty Image object and NOT hit the API.
// This is convenient for when you want to use the object for operations
// that do not require an initial GET request. To retrieve the image's details,
// run the following, which *will* call the API with a GET request:

$image->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

If, however, you *do* want to retrieve all the details of a remote image from the API, you just call:

```
$image->retrieve();
```

which will update the state of the local object. This gives you an element of control over your app's performance.

Delete an image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'    => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$image = $compute->getImage(['id' => '{imageId}']);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Retrieve metadata

This operation will retrieve the existing metadata for an image:

```
$metadata = $image->getMetadata();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Reset metadata

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$image = $compute->getImage(['id' => '{imageId}']);

$image->resetMetadata([
    'key' => 'value',
]);
```

This operation will _replace_ all existing metadata with whatever is provided in the request. Any existing metadata not specified in the request will be deleted.

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Merge metadata

This operation will _merge_ specified metadata with what already exists. Existing values will be overriden, new values will be added. Any existing keys that are not specified in the request will remain unaffected.

```
$image->mergeMetadata([
    'foo' => 'bar',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Retrieve image metadata item

This operation allows you to retrieve the value for a specific metadata item:

```
$itemValue = $image->getMetadataItem('key');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete image metadata item

This operation allows you to remove a specific metadata item:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$compute = $openstack->computeV2(['region' => '{region}']);

$image = $compute->getImage(['id' => '{imageId}']);

$image->deleteMetadataItem('key');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Server states

Servers contain a status attribute that indicates the current server state. You can filter on the server status when you complete a list servers request. The server status is returned in the response body. The server status is one of the following values:

State	Description
ACTIVE	The server is active.
BUILD-ING	The server has not finished the original build process.
DELETED	The server is permanently deleted.
ERROR	The server is in error.
HARD_REBOOT	The server is hard rebooting. This is equivalent to pulling the power plug on a physical server, plugging it back in, and rebooting it.
PASS-WORD	The password is being reset on the server.
PAUSED	In a paused state, the state of the server is stored in RAM. A paused server continues to run in frozen state.
REBOOT	The server is in a soft reboot state. A reboot command was passed to the operating system.
REBUILD	The server is currently being rebuilt from an image.
RESCUED	The server is in rescue mode. A rescue image is running with the original server image attached.
RESIZED	Server is performing the differential copy of data that changed during its initial copy. Server is down for this stage.
RE-VERT_RESIZE	The resize or migration of a server failed for some reason. The destination server is being cleaned up and the original source server is restarting.
SOFT_DELETED	The server is marked as deleted but the disk images are still available to restore.
STOPPED	The server is powered off and the disk image still persists.
SUS-PENDED	The server is suspended, either by request or necessity. This status appears for only the following hypervisors: XenServer/XCP, KVM, and ESXi. Administrative users may suspend an instance if it is infrequently used or to perform system maintenance. When you suspend an instance, its VM state is stored on disk, all memory is written to disk, and the virtual machine is stopped. Suspending an instance is similar to placing a device in hibernation; memory and vCPUs become available to create other instances.
UN-KNOWN	The state of the server is unknown. Contact your cloud provider.
VER-IFY_RESIZE	System is awaiting confirmation that the server is operational after a move or resize.

Identity v3

Credentials

Add credential

Create a secret/access pair for use with ec2 style auth. This operation will generates a new set of credentials that map the user/tenant pair.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
```

```
    'scope'    => ['project' => ['id' => '{projectId}']]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$credential = $identity->createCredential([
    'blob'      => '{blob}',
    'projectId' => '{projectId}',
    'type'      => '{type}',
    'userId'    => '{userId}'
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List credentials

List all credentials for a given user.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

foreach ($identity->listCredentials() as $credential) {
    /** @var $credential \OpenStack\Identity\v3\Models\Credential */
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show credential details

Retrieve a user's access/secret pair by the access key.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
```

```
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$credential = $identity->getCredential('credentialId');
$credential->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update credential

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$credential = $identity->getCredential('credentialId');

$credential->type = 'foo';
$credential->blob = 'bar';

$credential->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete credential

Delete a user's access/secret pair.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
```

```
        'password' => '{password}'
    ],
    'scope'    => ['project' => ['id' => '{projectId}']]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$credential = $identity->getCredential('credentialId');
$credential->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Domains

Add domain

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->createDomain([
    'description' => '{description}',
    'enabled'     => true,
    'name'        => '{name}'
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List domains

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
```

```
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

foreach ($identity->listDomains() as $domain) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show domain details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');
$domain->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update domain

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$domain->enabled = false;

$domain->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete domain

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');
$domain->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List roles for domain user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

foreach ($domain->listUserRoles(['userId' => '{domainUserId}']) as $role) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Grant role to domain user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$domain->grantUserRole([
    'userId' => '{domainUserId}',
```

```
    'roleId' => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Check role for domain user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$result = $domain->checkUserRole(['userId' => '{domainUserId}', 'roleId' => '{roleId}
↪']);

if (true === $result) {
    // It exists!
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Revoke role for domain user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
```

```
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$domain->revokeUserRole([
    'userId' => '{domainUserId}',
    'roleId' => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List roles for domain group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

foreach ($domain->listGroupRoles(['groupId' => '{groupId}']) as $role) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Grant role to domain group

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$domain->grantGroupRole([
    'groupId' => '{groupId}',
    'roleId'  => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Check role for domain group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$result = $domain->checkGroupRole(['groupId' => '{groupId}', 'roleId' => '{roleId}']);

if (true === $result) {
    // It exists!
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Revoke role for domain group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$domain = $identity->getDomain('{domainId}');

$domain->revokeGroupRole([
    'groupId' => '{groupId}',
    'roleId'  => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Endpoints

Add endpoints

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
```

```
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$endpoint = $identity->createEndpoint([
    'interface' => \OpenStack\Identity\v3\Enum::INTERFACE_INTERNAL,
    'name'      => '{endpointName}',
    'region'    => '{region}',
    'url'       => '{endpointUrl}',
    'serviceId' => '{serviceId}'
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Get endpoint

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$endpoint = $identity->getEndpoint('{endpointId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List endpoints

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
```

```
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

foreach ($identity->listEndpoints() as $endpoint) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update endpoint

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$endpoint = $identity->getEndpoint('{endpointId}');

$endpoint->interface = \OpenStack\Identity\v3\Enum::INTERFACE_PUBLIC;

$endpoint->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete endpoint

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$endpoint = $identity->getEndpoint('{endpointId}');
$endpoint->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Groups

Add group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->createGroup([
    'description' => '{description}',
    'name'        => '{name}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List groups

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

foreach ($identity->listGroups() as $group) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show group details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');
$group->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');

$group->description = 'foo';
$group->name = 'bar';

$group->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');
```

```
$group->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List users in a group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');

foreach ($group->listUsers() as $user) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Add user to group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');

$group->addUser(['userId' => '{groupUserId}']);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Remove user from group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');

$group->removeUser(['userId' => '{groupUserId}']);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Check user membership in a group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
```

```
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$group = $identity->getGroup('{groupId}');

$result = $group->checkMembership(['userId' => '{groupUserId}']);

if (true === $result) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Policies

Warning: Due to lack of information in upstream documentation, we are unable to offer code samples for this resource yet.

Add policy

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List policies

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show policy details

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update policy

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete policy

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Projects

Add project

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->createProject([
    'description' => '{description}',
    'enabled'     => true,
    'name'        => '{name}'
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List projects

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);
```

```
foreach ($identity->listProjects() as $project) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show project details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');
$project->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update project

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$project->enabled = false;

$project->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete project

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$project->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List roles for project user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
```

```
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

foreach ($project->listUserRoles(['userId' => '{projectUserId}']) as $role) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Grant role to project user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$project->grantUserRole([
    'userId' => '{projectUserId}',
    'roleId' => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Check role for project user

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$result = $project->checkUserRole([
    'userId' => '{projectUserId}',
    'roleId' => '{roleId}',
]);

if (true === $result) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Revoke role for project user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$project->revokeUserRole([
    'userId' => '{projectUserId}',
```

```
    'roleId' => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List roles for project group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

foreach ($project->listGroupRoles(['groupId' => '{groupId}']) as $role) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Grant role to project group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$project->grantUserRole([
    'userId' => '{projectUserId}',
    'roleId' => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Check role for project group

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$result = $project->checkGroupRole([
    'groupId' => '{groupId}',
    'roleId'  => '{roleId}',
]);

if (true === $result) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Revoke role for project group

```
<?php

require 'vendor/autoload.php';
```

```
$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$project = $identity->getProject('{id}');

$project->revokeGroupRole([
    'groupId' => '{groupId}',
    'roleId'  => '{roleId}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Roles

Add role

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

$role = $identity->createRole([
    'name' => '{name}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List roles

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

foreach ($identity->listRoles() as $role) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List role assignments

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

foreach ($identity->listRoleAssignments() as $assignment) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Services

Add service

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$service = $identity->createService([
    'name' => '{serviceName}',
    'type' => '{serviceType}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List services

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

foreach ($identity->listServices() as $service) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show service details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$service = $identity->getService('{serviceId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete service

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$service = $identity->getService('{serviceId}');
$service->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Tokens

Authenticate (generate) token

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Generate token with user ID

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

```
$identity = $openstack->identityV3();

$token = $identity->generateToken([
    'user' => [
        'id'       => '{userId}',
        'password' => '{password}'
    ]
]);
```

Generate token with username

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
```

```
        'name'     => '{username}',
        'password' => '{password}',
        'domain'   => ['id' => '{domainId}']
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

```
$identity = $openstack->identityV3();

// Since usernames will not be unique across an entire OpenStack installation,
// when authenticating with them you must also provide your domain ID. You do
// not have to do this if you authenticate with a user ID.

$token = $identity->generateToken([
    'user' => [
        'name'     => '{username}',
        'password' => '{password}',
        'domain'   => [
            'id' => '{domainId}'
        ]
    ]
]);
```

Generate token from ID

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

$token = $identity->generateToken([
    'tokenId' => '{tokenId}',
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

Generate token scoped to project ID

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

```
$identity = $openstack->identityV3();

$token = $identity->generateToken([
    'user' => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

Generate token scoped to project name

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

```
$identity = $openstack->identityV3();

// Since project names will not be unique across an entire OpenStack installation,
// when authenticating with them you must also provide your domain ID. You do
// not have to do this if you authenticate with a project ID.

$token = $identity->generateToken([
```

```
    'user' => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'name' => '{projectName}',
            'domain' => [
                'id' => '{domainId}'
            ]
        ]
    ]
]);
```

Validate token

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'name'     => '{username}',
        'password' => '{password}',
        'domain'   => ['id' => '{domainId}']
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$result = $identity->validateToken('{tokenId}');

if (true === $result) {
    // It's valid!
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Revoke token

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
```

```
        'name'     => '{username}',
        'password' => '{password}',
        'domain'   => ['id' => '{domainId}']
    ],
    'scope' => [
        'project' => ['id' => '{projectId}']
    ]
]);
```

```
$identity = $openstack->identityV3(['region' => '{region}']);

$identity->revokeToken('{tokenId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Users

Add user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$identity = $openstack->identityV3();

$user = $identity->createUser([
    'defaultProjectId' => '{defaultProjectId}',
    'description'      => '{description}',
    'domainId'         => '{domainId}',
    'email'            => '{email}',
    'enabled'          => true,
    'name'             => '{name}',
    'password'         => '{userPass}'
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List users

```
<?php
```

```
require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

foreach ($identity->listUsers() as $user) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Show user details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

$user = $identity->getUser('{id}');
$user->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

$user = $identity->getUser('{id}');

$user->description = '{description}';
$user->name = '{name}';

$user->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

$user = $identity->getUser('{id}');
$user->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List groups for user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();

$user = $identity->getUser('{id}');

foreach ($user->listGroups() as $group) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List projects for user

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$identity = $openstack->identityV3();
```

```
$user = $identity->getUser('{id}');

foreach ($user->listProjects() as $project) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Images v2

Images

Create image

The only required attribute when creating a new image is `name`.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

$image = $service->createImage([
    'name'            => '{name}',
    'tags'            => ['{tag1}', '{tag2}'],
    'containerFormat' => '{containerFormat}',
    'diskFormat'      => '{diskFormat}',
    'visibility'      => '{visibility}',
    'minDisk'         => 10,
    'protected'       => true,
    'minRam'          => 10,
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List images

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
```

```
    'scope'    => ['project' => ['id' => '{projectId}']]
]);
```

```
$images = $openstack->imagesV2()
                    ->listImages();

foreach ($images as $image) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Show image details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

$image = $service->getImage('{imageId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

```

```
$image = $service->getImage('{imageId}');
$image->update([
    'minDisk'    => 1,
    'minRam'     => 1,
    'name'       => '{name}',
    'protected'  => false,
    'visibility' => '{visibility}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->imagesV2()
    ->getImage('{imageId}')
    ->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Reactivate image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

$image = $service->getImage('{imageId}');
$image->reactivate();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Deactivate image

If you try to download a deactivated image, a Forbidden error is returned.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

$image = $service->getImage('{imageId}');
$image->deactivate();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Upload binary data

Before you can store binary image data, you must meet the following preconditions:

- The image must exist.
- You must set the disk and container formats in the image.
- The image status must be `queued`.
- Your image storage quota must be sufficient.

The size of the data that you want to store must not exceed the size that the Image service allows.

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

$image  = $service->getImage('{imageId}');
$stream = \GuzzleHttp\Psr7\stream_for(fopen('{fileName}', 'r'));
$image->uploadData($stream);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Download binary data

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->imagesV2();

$image  = $service->getImage('{imageId}');

/** @var \GuzzleHttp\Psr7\Stream $stream */
$stream = $image->downloadData();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Image Members

Add member to image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$member = $openstack->imagesV2()
                    ->getImage('{imageId}')
                    ->addMember('{tenantId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

List image members

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
```

```
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$image = $openstack->imagesV2()
                   ->getImage('{imageId}');

foreach ($image->listMembers() as $member) {
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Show member details

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$member = $openstack->imagesV2()
                    ->getImage('{imageId}')
                    ->getMember('{tenantId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Remove member from image

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->imagesV2()
          ->getImage('{imageId}')
          ->getMember('{tenantId}')
          ->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update status of image member

```
<?php

require 'vendor/autoload.php';

use OpenStack\Images\v2\Models\Member;

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->imagesV2()
    ->getImage('{imageId}')
    ->getMember('{tenantId}')
    ->updateStatus(Member::STATUS_ACCEPTED);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Networking v2

Networks

Create network

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => [
        'project' => [
            'id' => '{projectId}'
        ]
```

```
    ]
]);
```

```
$networking = $openstack->networkingV2();

$options = [
    'name'         => '{networkName}',
    'adminStateUp' => true,
];

// Create the network
$network = $networking->createNetwork($options);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Create networks

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$options = [
    [
      'name' => '{networkName1}'
    ],
    [
      'name' => '{networkName2}'
    ],
];

$networks = $networking->createNetworks($options);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Get network

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$network = $networking->getNetwork('{networkId}');

// By default, this will return an empty Network object and NOT hit the API.
// This is convenient for when you want to use the object for operations
// that do not require an initial GET request. To retrieve the network's details,
// run the following, which *will* call the API with a GET request:

$network->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update network

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();
```

```
$network = $networking->getNetwork('{networkId}');

$network->name = '{newName}';
$network->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete network

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$network = $networking->getNetwork('{networkId}');

$network->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Subnets

Create subnet

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => ['project' => ['id' => '{projectId}']]
]);
```

```
$networking = $openstack->networkingV2();

$options = [
    'name'      => '{subnetName}',
    'networkId' => '{networkId}',
    'ipVersion' => 4,
    'cidr'      => '192.168.199.0/24'
];

// Create the subnet
$subnet = $networking->createSubnet($options);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

To create a subnet with gateway IP:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$options = [
    'name'      => 'My subnet',
    'networkId' => '{networkId}',
    'ipVersion' => 4,
    'cidr'      => '192.168.199.0/25',
    'gatewayIp' => '192.168.199.128'
];

// Create the subnet
$subnet = $networking->createSubnet($options);
```

To create a subnet with host routes:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
```

```
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$options = [
    'name'       => 'My subnet',
    'networkId'  => '{networkId}',
    'ipVersion'  => 4,
    'cidr'       => '192.168.199.0/24',
    'hostRoutes' => [[
        'destination' => '1.1.1.0/24',
        'nexthop'     => '192.168.19.20'
    ]]
];

// Create the subnet
$subnet = $networking->createSubnet($options);
```

Get subnet

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$subnet = $networking->getSubnet('{subnetId}');

// By default, this will return an empty Subnet object and NOT hit the API.
// This is convenient for when you want to use the object for operations
// that do not require an initial GET request. To retrieve the subnet's details,
// run the following, which *will* call the API with a GET request:
```

```
$subnet->retrieve();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update subnet

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$subnet = $networking->getSubnet('{subnetId}');

$subnet->name = '{newName}';
$subnet->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete subnet

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope' => [
        'project' => [
            'id' => '{projectId}'
        ]
    ]
]);
```

```
$networking = $openstack->networkingV2();

$subnet = $networking->getSubnet('{subnetId}');

$subnet->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Ports

Create port

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$networking = $openstack->networkingV2();

$port = $networking->createPort([
    'name'         => 'portName',
    'networkId'    => '{networkId}',
    'adminStateUp' => true
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Create ports

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$networking = $openstack->networkingV2();

$ports = $networking->createPorts([
    [
        'name'         => 'port1',
```

```
        'networkId'    => '{networkId}',
        'adminStateUp' => true
    ],
    [
        'name'         => 'port2',
        'networkId'    => '{networkId}',
        'adminStateUp' => true
    ],
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Get port

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$networking = $openstack->networkingV2();

$port = $networking->getPort('{portId}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Update port

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$networking = $openstack->networkingV2();

$port = $networking->getPort('{portId}');
$port->name = 'newName';
$port->update();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete port

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => ['id' => '{userId}', 'password' => '{password}'],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$networking = $openstack->networkingV2();

$port = $networking->getPort('{portId}');
$port->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Object Store v1

Account

Show account details

To work with an Object Store account, you must first retrieve an account object like so:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$account = $openstack->objectStoreV1()
                     ->getAccount();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Get account metadata

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->objectStoreV1();

$account = $service->getAccount();
$metadata = $account->getMetadata();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Replace all metadata with new values

In order to replace all existing metadata with a set of new values, you can use this operation. Any existing metadata items which not specified in the new set will be removed. For example, say an account has the following metadata already set:

```
Foo: value1
Bar: value2
```

and you *reset* the metadata with these values:

```
Foo: value4
Baz: value3
```

the metadata of the account will now be:

```
Foo: value4
Baz: value3
```

To merge metadata, you must run:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->objectStoreV1();

$account = $service->getAccount();

$account->resetMetadata([
    '{key_1}' => '{val_1}',
    '{key_2}' => '{val_2}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Merge new metadata values with existing

In order to merge a set of new metadata values with the existing metadata set, you can use this operation. Any existing metadata items which are not specified in the new set will be preserved. For example, say an account has the following metadata already set:

```
Foo: value1
Bar: value2
```

and you merge them with these values:

```
Foo: value4
Baz: value3
```

the metadata of the account will now be:

```
Foo: value4
Bar: value2
Baz: value3
```

To reset metadata, you must run:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->objectStoreV1();

$account = $service->getAccount();

$account->mergeMetadata([
    '{key_1}' => '{val_1}',
    '{key_2}' => '{val_2}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Containers

Show details for a container

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$container = $openstack->objectStoreV1()
                       ->getContainer('{containerName}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

At this point, the object returned is *empty* because we did not execute a HTTP request to receive the state of the container from the API. This is in accordance with one of the SDK's general policies of not assuming too much at the expense of performance.

To synchronize the local object's state with the remote API, you can run:

```
$container->retrieve();

printf("%s container has %d objects and %d bytes",
    $container->name, $container->objectCount, $container->bytesUsed);
```

and all of the local properties will match those of the remote resource.

List containers

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->objectStoreV1();

foreach ($service->listContainers() as $container) {
    /** @var $container \OpenStack\ObjectStore\v1\Models\Container */
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

When listing containers, you must be aware that not *all* information about a container is returned in a collection. Very often only the object count, bytes used and container name will be exposed. If you would like to access all of the remote state of a collection item, you can call `retrieve` like so:

```
foreach ($containers as $container) {
    $container->retrieve();
}
```

If you have a large collection of containers, this will slow things down because you're issuing a HEAD request per container.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Delete container

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->objectStoreV1()
          ->getContainer('{containerName}')
          ->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

The API will only accept DELETE requests on containers when they are empty. If you have a container with any objects inside, the operation will fail.

Get metadata

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
/** @var array $metadata */
$metadata = $openstack->objectStoreV1()
                      ->getContainer('{containerName}')
                      ->getMetadata();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

The returned value will be a standard associative array, or hash, containing arbitrary key/value pairs. These will correspond to the values set either when the container was created, or when a previous `mergeMetadata` or `resetMetadata` operation was called.

Replace all metadata with new values

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$service = $openstack->objectStoreV1();

$container = $service->getContainer('{containerName}');

$container->resetMetadata([
    '{key_1}' => '{val_1}',
    '{key_2}' => '{val_2}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

In order to replace all existing metadata with a set of new values, you can use this operation. Any existing metadata items which not specified in the new set will be removed. For example, say an account has the following metadata already set:

```
Foo: value1
Bar: value2
```

and you *reset* the metadata with these values:

```
Foo: value4
Baz: value3
```

the metadata of the account will now be:

```
Foo: value4
Baz: value3
```

Merge new metadata values with existing

```php
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```php
$service = $openstack->objectStoreV1();

$container = $service->getContainer('{containerName}');

$container->mergeMetadata([
    '{key_1}' => '{val_1}',
    '{key_2}' => '{val_2}',
]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

In order to merge a set of new metadata values with the existing metadata set, you can use this operation. Any existing metadata items which are not specified in the new set will be preserved. For example, say an account has the following metadata already set:

```
Foo: value1
Bar: value2
```

and you merge them with these values:

```
Foo: value4
Baz: value3
```

the metadata of the account will now be:

```
Foo: value4
Bar: value2
Baz: value3
```

Objects

Show details for an object

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
/** @var \OpenStack\ObjectStore\v1\Models\Object $object */
$object = $openstack->objectStoreV1()
                    ->getContainer('{containerName}')
                    ->getObject('{objectName}');
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

At this point, the object returned is *empty* because we did not execute a HTTP request to receive the state of the container from the API. This is in accordance with one of the SDK's general policies of not assuming too much at the expense of performance.

To synchronize the local object's state with the remote API, you can run:

```
$object->retrieve();

printf("%s/%s is %d bytes long and was last modified on %s",
    $object->containerName, $object->name, $object->contentLength, $object->
↪lastModified);
```

and all of the local properties will match those of the remote resource. The `retrieve` call, although fetching all of the object's metadata, will not download the object's content. To do this, see the next section.

Download an object

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
```

```
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
/** @var \GuzzleHttp\Stream\Stream $stream */
$stream = $openstack->objectStoreV1()
                    ->getContainer('{containerName}')
                    ->getObject('{objectName}')
                    ->download();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

As you will notice, a Stream object is returned by this call. For more information about dealing with streams, please consult Guzzle's docs.

List objects

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$container = $openstack->objectStoreV1()
                       ->getContainer('{containerName}');

foreach ($container->listObjects() as $object) {
    /** @var \OpenStack\ObjectStore\v1\Models\Object $object */
}
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

When listing objects, you must be aware that not *all* information about a container is returned in a collection. Very often only the MD5 hash, last modified date, bytes used, content type and object name will be returned. If you would like to access all of the remote state of a collection item, you can call `retrieve` like so:

```
foreach ($objects as $object) {
    // To retrieve metadata
    $object->retrieve();
}
```

If you have a large collection of $object, this will slow things down because you're issuing a HEAD request per object.

By default, PHP generators are used to represent collections of resources in the SDK. The benefit of using generators is that it generally improves performance, since objects are not saved in memory as the iteration cycle goes on; instead, each resource is directly output to the user-defined `foreach` loop. For all intents and purposes, you interact with generators like any other Traversable object, but to retain collections in memory, you will need to implement your own logic.

Create an object

When creating an object, you can upload its content according to a string representation:

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$options = [
    'name'    => '{objectName}',
    'content' => '{objectContent}',
];

/** @var \OpenStack\ObjectStore\v1\Models\Object $object */
$object = $openstack->objectStoreV1()
                    ->getContainer('{containerName}')
                    ->createObject($options);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

If that is not optimal or convenient, you can use a stream instead. Any instance of `\Psr\Http\Message\StreamInterface` is acceptable. For example, to use a normal Guzzle stream:

```
<?php

require 'vendor/autoload.php';

use GuzzleHttp\Psr7\Stream;

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
// You can use any instance of \Psr\Http\Message\StreamInterface
$stream = new Stream(fopen('/path/to/object.txt', 'r'));

$options = [
    'name'   => '{objectName}',
    'stream' => $stream,
];

/** @var \OpenStack\ObjectStore\v1\Models\Object $object */
$object = $openstack->objectStoreV1()
    ->getContainer('{containerName}')
    ->createObject($options);
```

Create a large object (over 5GB)

For large objects (those over 5GB), you will need to use a concept in Swift called Dynamic Large Objects (DLO). When uploading, this is what happens under the hood:

1. The large file is separated into smaller segments
2. Each segment is uploaded
3. A manifest file is created which, when requested by clients, will concatenate all the segments as a single file

To upload a DLO, you need to call:

```
<?php

require 'vendor/autoload.php';

use Guzzle\Stream\Stream;

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$options = [
    'name'   => 'object_name.txt',
    'stream' => new Stream(fopen('/path/to/large_object.mov', 'r')),
];

// optional: specify the size of each segment in bytes
$options['segmentSize'] = 1073741824;

// optional: specify the container where the segments live. This does not necessarily␣
↪have to be the
// same as the container which holds the manifest file
$options['segmentContainer'] = 'test_segments';


/** @var \OpenStack\ObjectStore\v1\Models\Object $object */
```

```
$object = $openstack->objectStoreV1()
                    ->getContainer('test')
                    ->createLargeObject($options);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Copy object

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->objectStoreV1()
          ->getContainer('{containerName}')
          ->getObject('{objectName}')
          ->copy([
              'destination' => '{newContainerName}/{newObjectName}'
          ]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Delete object

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->objectStoreV1()
          ->getContainer('{containerName}')
          ->getObject('{objectName}')
          ->delete();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

Get metadata

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
/** @var array $metadata */
$metadata = $openstack->objectStoreV1()
                      ->getContainer('{containerName}')
                      ->getObject('{objectName}')
                      ->getMetadata();
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

The returned value will be a standard associative array, or hash, containing arbitrary key/value pairs. These will correspond to the values set either when the object was created, or when a previous `mergeMetadata` or `resetMetadata` operation was called.

Replace all metadata with new values

```
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```
$openstack->objectStoreV1()
          ->getContainer('{containerName}')
          ->getObject('{objectName}')
          ->resetMetadata([
              '{key_1}' => '{val_1}',
              '{key_2}' => '{val_2}',
          ]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

In order to replace all existing metadata with a set of new values, you can use this operation. Any existing metadata items which not specified in the new set will be removed. For example, say an account has the following metadata already set:

```
Foo: value1
Bar: value2
```

and you *reset* the metadata with these values:

```
Foo: value4
Baz: value3
```

the metadata of the account will now be:

```
Foo: value4
Baz: value3
```

Merge new metadata values with existing

```php
<?php

require 'vendor/autoload.php';

$openstack = new OpenStack\OpenStack([
    'authUrl' => '{authUrl}',
    'region'  => '{region}',
    'user'    => [
        'id'       => '{userId}',
        'password' => '{password}'
    ],
    'scope'   => ['project' => ['id' => '{projectId}']]
]);
```

```php
$openstack->objectStoreV1()
          ->getContainer('{containerName}')
          ->getObject('{objectName}')
          ->mergeMetadata([
              '{key_1}' => '{val_1}',
              '{key_2}' => '{val_2}',
          ]);
```

To see all the required and optional parameters for this operation, along with their types and descriptions, view the reference documentation.

In order to merge a set of new metadata values with the existing metadata set, you can use this operation. Any existing metadata items which are not specified in the new set will be preserved. For example, say an account has the following metadata already set:

```
Foo: value1
Bar: value2
```

and you merge them with these values:

```
Foo: value4
Baz: value3
```

the metadata of the account will now be:

```
Foo: value4
Bar: value2
Baz: value3
```

CHAPTER 4

Help and support

If you have specific problems or bugs with this SDK, please file an issue on our official Github repo. We also have a mailing list, so feel free to join to keep up to date with all the latest changes and announcements to the library.

For general feedback and support requests, send an email to sdk-support@rackspace.com.

You can also find assistance via IRC on #rackspace at freenode.net.

CHAPTER 5

Contributing

If you'd like to contribute to the project, or require help running the unit/integration tests, please view the contributing guidelines.