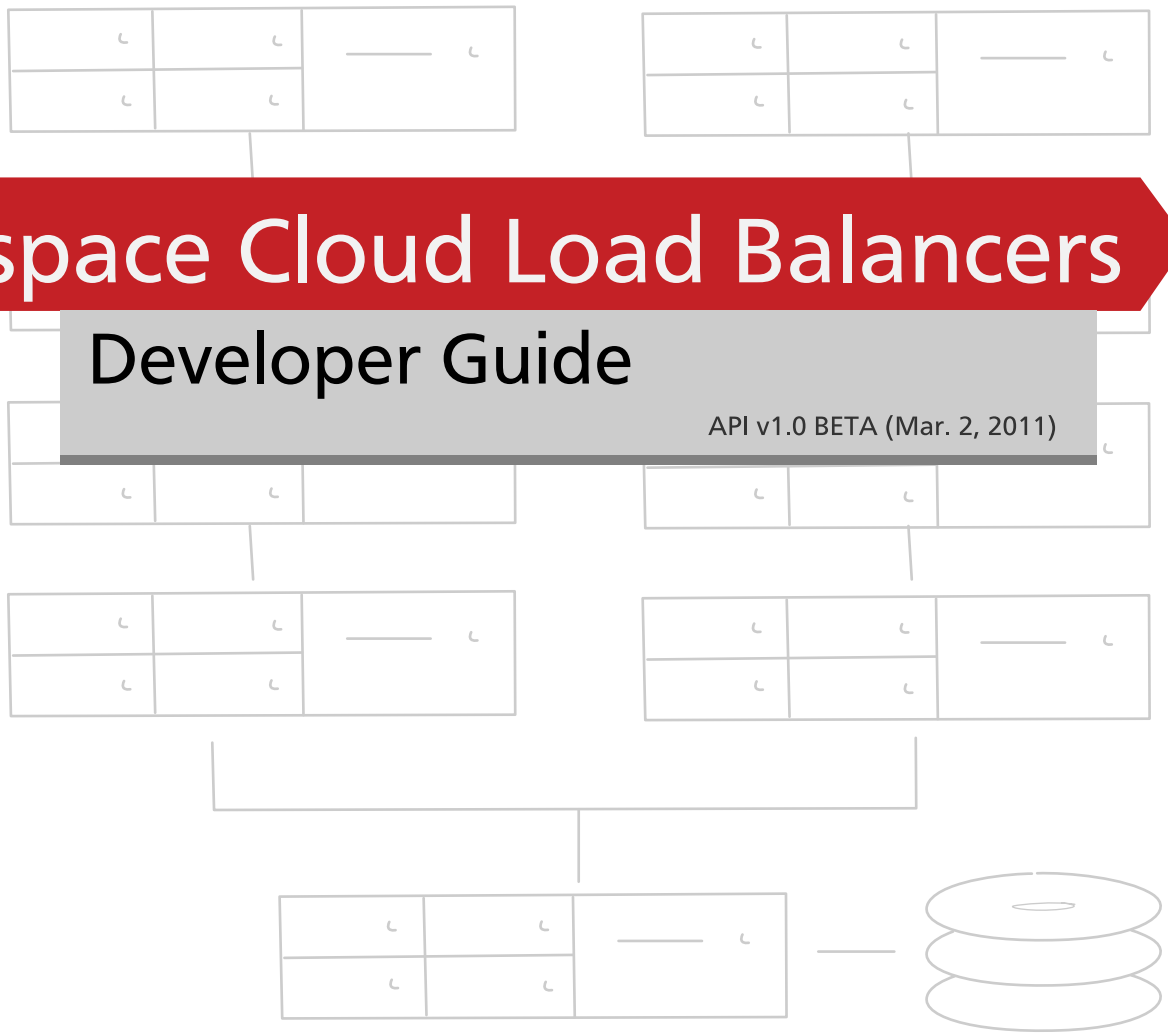
Rackspace Cloud Load Balancers

Developer Guide

API v1.0 BETA (Mar. 2, 2011)



rackspace® HOSTING

docs.rackspacecloud.com/api

Rackspace Cloud Load Balancers Developer Guide

API v1.0 BETA (03/02/11)

This document is intended for software developers interested in developing applications using the Rackspace Cloud Load Balancers Application Program Interface (API). The document is for informational purposes only and is provided "AS IS."

Table of Contents

1. Overview 1
1.1. Intended Audience 1
1.2. Document Change History 2
1.3. Additional Resources 2
2. Concepts 3
2.1. Load Balancer 3
2.2. Virtual IP 3
2.3. Node 3
2.4. Health Monitor 3
2.4.1. Passive Health Monitor 3
2.4.2. Active Health Monitor 4
2.5. Session Persistence 4
2.6. Connection Logging 4
3. General API Information 5
3.1. Authentication 5
3.2. Service Access / Endpoints 6
3.3. Request/Response Types 7
3.4. Content Compression 7
3.5. Persistent Connections 7
3.6. Paginated Collections 8
3.7. Efficient Polling with the `Changes-Since` Parameter 8
3.8. Limits 8
3.8.1. Rate Limits 8
3.8.2. Absolute Limits 9
3.8.3. Determining Limits Programmatically 9
4. API Operations 11
4.1. List Load Balancers 11
4.2. List Load Balancer Details 13
4.3. Create Load Balancer 16
4.4. Remove Load Balancer 20
4.5. List, Add, Modify, and Remove Nodes 20
4.6. Update Load Balancer Attributes 24
4.7. Virtual IP Management 25
4.8. Usage Reporting 26
4.9. Access List Management 30
4.10. Active Health Monitoring 32
4.10.1. Connection Monitor 34
4.10.2. HTTP/HTTPS Monitor 35
4.11. Session Persistence 35
4.12. Connection Logging 36
4.13. Connection Throttling 37
4.14. Load Balancing Protocols 39
4.15. Load Balancing Algorithms 41
4.16. Load Balancer Status 43
4.17. Node Condition 44
5. API Faults 45
5.1. `serviceFault` 45
5.2. `loadBalancerFault` 45

5.3. badRequest 45
5.4. itemNotFound 46
5.5. overLimit 46
5.6. unauthorized 46
5.7. outOfVirtualIps 46
5.8. immutableEntity 47
5.9. unprocessableEntity 47
5.10. serviceUnavailable 47

List of Tables

3.1. Regionalized Service Endpoints 6
3.2. JSON and XML Response Formats 7
3.3. Encoding Headers 7
3.4. Default Rate Limits 9
4.1. Virtual IP Types 25
4.2. Health Monitor Types 33
4.3. Session Persistence Modes 35
4.4. Load Balancing Algorithms 41
4.5. Load Balancer Statuses 43
4.6. Load Balancer Node Conditions 44

List of Examples

3.1. Authentication Request 5
3.2. Authentication Response 6
3.3. List Limits Response: XML 10
3.4. List Limits Response: JSON 10
4.1. List Load Balancers Response: XML 11
4.2. List Load Balancers Response: JSON 12
4.3. List Load Balancer Details Request: XML 14
4.4. List Load Balancers Details Response: JSON 15
4.5. Create Load Balancer (Required Attributes) Request: XML 16
4.6. Create Load Balancer (Required Attributes) Request: JSON 17
4.7. Create Load Balancer (Required Attributes with Shared IP) Request: XML 17
4.8. Create Load Balancer (Required Attributes with Shared IP) Request: JSON 18
4.9. Create Load Balancer (Required Attributes with Shared IP) Response: XML 19
4.10. List Node Response: XML 21
4.11. Add Node Request: XML 21
4.12. Add Node Response: XML 22
4.13. Add Node Request: JSON 22
4.14. Add Node Response: JSON 22
4.15. Update Node Attributes Request: XML 22
4.16. Update Node Attributes Request: JSON 23
4.17. Update Load Balancer Attributes Request: XML 24
4.18. Update Load Balancer Attributes Request: JSON 25
4.19. List Virtual IPs Response: XML 26
4.20. List Virtual IPs Response: JSON 26
4.21. Report Load Balancer Usage Response: XML 27
4.22. Report Load Balancer Usage Response: JSON 28
4.23. Report Account Billing Response: XML 29
4.24. Retrieve Access List Response: XML 30
4.25. Retrieve Access List Response: JSON 31
4.26. Update Access List Attributes Request: XML 32
4.27. Update Access List Attributes Request: JSON 32
4.28. Monitor Connections Response: XML 33
4.29. Monitor Connections Response: JSON 33
4.30. Monitor HTTP Response: XML 33
4.31. Monitor HTTPs Response: XML 34
4.32. Monitor Connections Request: XML 34
4.33. Monitor Connections Request: JSON 34
4.34. List Session Persistence Configuration Response: XML 36
4.35. List Session Persistence Configuration Response: JSON 36
4.36. Set Session Persistence Type Request: XML 36
4.37. Set Session Persistence Type Request: JSON 36
4.38. List Connection Logging Configuration Response: XML 37
4.39. List Connection Logging Configuration Response: JSON 37
4.40. Enable Connection Logging Request: XML 37
4.41. Enable Connection Logging Request: JSON 37
4.42. List Connection Throttling Configuration Response: XML 38
4.43. List Connection Throttling Configuration Response: JSON 38
4.44. Update Connection Throttling Configuration Request: XML 38

4.45. Update Connection Throttling Configuration Request: JSON 39
4.46. List Load Balancing Protocols Response: XML 39
4.47. List Load Balancing Protocols Response: JSON 40
4.48. List Load Balancing Algorithms Response: XML 41
4.49. List Load Balancing Algorithms Response: JSON 42

1. Overview

1.1. Intended Audience

This guide is intended for software developers who want to create applications using the Rackspace Cloud Load Balancers API. It assumes the reader has a general understanding of load balancing concepts and is familiar with:

- RESTful web services
- HTTP/1.1 conventions
- JSON and/or XML serialization formats
- Atom syndication format

1.2. Document Change History

This version of the Developer Guide replaces and supersedes all previous versions. The most recent changes are described in the table below:

Revision Date	Summary of Changes
Mar. 2, 2011	• Revised code samples and formatting to address initial beta feedback.
Feb. 23, 2011	• Initial release for public beta.

1.3. Additional Resources

You can download the most current version of this and other API-related documents, including this document, from http://docs.rackspacecloud.com/api/.

For more details about Rackspace Cloud products, refer tohttp://www.rackspace.com/cloud. This site also offers links to Rackspace's official support channels, including knowledge base articles, forums, phone, chat, and email.

You can also follow updates and announcements via twitter at http://www.twitter.com/rackcloud

2. Concepts

To use the Rackspace Cloud Load Balancers API effectively, you should understand several key concepts:

2.1. Load Balancer

A load balancer is a logical device which belongs to a cloud account. It is used to distribute workloads between multiple back-end systems or services, based on the criteria defined as part of its configuration.

2.2. Virtual IP

A virtual IP is an IP address configured on the load balancer for use by clients connecting to a service that is load balanced. Incoming connections are distributed to back-end nodes based on the configuration of the load balancer.

2.3. Node

A node is a back-end device providing a service on a specified IP and port.

2.4. Health Monitor

A health monitor is a feature of the load balancer that is used to determine whether or not a back-end node is usable for processing a request. The service supports two types of health monitors: passive and active.

2.4.1. Passive Health Monitor

By default, all load balancing configurations utilize a passive health monitor, which is a technique that analyzes the response of the back-end server to determine whether or not it is capable of processing a request. Passive health monitoring verifies the condition of a node using the following criteria:

- The connection is refused by the back-end server or the connection takes longer than 4 seconds to be established.
- The connection is closed prematurely or the beginning of the response is not received within 30 seconds.
- HTTP: An invalid HTTP response is received or a 503 (Service Unavailable) response code is received.
- SSL Passthrough: The SSL handshake to the back-end server fails.

If these tests fail, then the service will attempt to locate another configured node that is capable of supporting the request until every node is exhausted. If this occurs, the requestor will receive a 503 (Service Unavailable) response for HTTP traffic or the connection will be dropped.

Note

If a given node fails a passive health monitor check three consecutive times, then the service assumes the node has failed and will place it in `OFFLINE` status. The service will not attempt to send any new requests to the node for at least 60 seconds, after which the service will speculatively begin sending requests to occasionally probe it to determine whether or not it has recovered.

2.4.2. Active Health Monitor

Active health monitoring is a technique that uses synthetic transactions that are executed at periodic intervals to determine the condition of a node. One of the advantages of active health monitoring is that it does not require active transactions to be processed by the load balancer to determine whether or not a node is suitable for handling traffic. When both active and passive monitoring are enabled, the decisions made by the active monitor override inferences made by the passive monitoring system.

The active health monitor can use one of three types of probes:

- `connect`
- `HTTP`
- `HTTPS`

These probes are executed at configured intervals; in the event of a failure, the node status changes to `OFFLINE` and the node will not receive traffic. If, after running a subsequent test, the probe detects that the node has recovered, then the node's status is changed to `ONLINE` and it is capable of servicing requests.

2.5. Session Persistence

Session persistence is a feature of the load balancing service that attempts to force subsequent connections to a service to be redirected to the same node as long as it is online.

2.6. Connection Logging

The connection logging feature allows for Apache-style access logs (for HTTP-based protocol traffic) or connection and transfer logging (for all other traffic) to a Cloud Files account. Logs are delivered to the account every hour.

3. General API Information

The Rackspace Cloud Load Balancers API is implemented using a ReSTful web service interface. Like other products in the Rackspace Cloud suite, the load balancing service shares a common token-based authentication system that allows seamless access between products and services.

Note

All requests to authenticate and operate the service are performed using HTTPS on TCP port 443.

3.1. Authentication

Every ReST request against the load balancing service requires the inclusion of a specific authorization token, supplied by the `X-Auth-Token` HTTP header. Customers obtain this token by first using the Rackspace Cloud Authentication Service and supplying a valid username and API access key.

Request

The Rackspace Cloud Authentication Service serves as the entry point to all Rackspace Cloud APIs and is itself a ReSTful web service. It is accessible at https://auth.api.rackspacecloud.com/v1.0. To authenticate, you must supply your username and API access key in x-headers:

- Use your Rackspace Cloud username as the username for the API. Place it in the `X-Auth-User` x-header.
- Obtain your API access key from the Rackspace Cloud Control Panel in the Your Account | API Access section. Place it in the `X-Auth-Key` x-header.

Example 3.1. Authentication Request

```
GET /v1.0 HTTP/1.1
Host: auth.api.rackspacecloud.com
X-Auth-User: jdoe
X-Auth-Key: a86850deb2742ec3cb41518e26aa2d89
```

Response

If authentication is successful, an HTTP status 204 (No Content) is returned with an `X-Auth-Token` (along with other Cloud Service headers that are not applicable to the load balancing service). An HTTP status of 401 (Unauthorized) is returned if authentication fails. All operations against the load balancing service must include the `X-Auth-Token` header.

Example 3.2. Authentication Response

```
HTTP/1.1 204 No Content
Date: Mon, 12 Nov 2007 15:32:21 GMT
Server: Apache
X-Server-Management-Url: https://servers.api.rackspacecloud.com/
v1.0/1234
X-Storage-Url: https://storage.clouddrive.com/v1/
CloudFS_9c83b-5ed4
X-CDN-Management-Url: https://cdn.clouddrive.com/v1/
CloudFS_9c83b-5ed4
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

Authentication tokens are typically valid for 24 hours. Applications should be designed to re-authenticate after receiving a 4 401 (Unauthorized) response from a service endpoint.

Note

Currently, service endpoints are not published in the response of the account number, which is a required element of the service endpoints.

3.2. Service Access / Endpoints

The load balancing service is a regionalized service, which allows for the caller to select a region into which a load balancer is to be provisioned.

To determine which region to operate against, select an endpoint from below. Replace the sample account ID number, 1234, with your actual account number returned as part of the authentication service response.

Table 3.1. Regionalized Service Endpoints

Region	Endpoint
Chicago (ORD)	`https://ord.loadbalancers.api.rackspacecloud.com/v1.0/1234/`
Dallas/Ft. Worth (DFW)	`https://dfw.loadbalancers.api.rackspacecloud.com/v1.0/1234/`

If load balancing Cloud Servers, you can determine the appropriate region to select by viewing your Cloud Servers list and creating a load balancer within the same region as the data center in which your Cloud Server resides. When your resources reside in the same region as your load balancer, it ensures your devices are in close proximity to each other and allows you to take advantage of ServiceNet connectivity for free data transfer between services.

If load balancing external servers, you can determine the appropriate region to select by choosing the region that is geographically as close to your external servers as possible.

3.3. Request/Response Types

The Rackspace Cloud Load Balancers API supports both the JSON and XML data serialization formats. The request format is specified using the `Content-Type` header and is required for operations that have a request body. The response format can be specified in requests using either the `Accept` header or adding an `.xml` or `.json` extension to the request URI. Note that it is possible for a response to be serialized using a format different from the request. If no response format is specified, JSON is the default. If conflicting formats are specified using both an `Accept` header and a query extension, the query extension takes precedence.

Some operations support an Atom representation that can be used to efficiently determine when the state of services has changed.

Table 3.2. JSON and XML Response Formats

Format	Accept Header	Query Extension	Default
JSON	application/json	.json	Yes
XML	application/xml	.xml	No
Atom	application/atom +xml	.atom	No

3.4. Content Compression

Request and response body data may be encoded with gzip compression in order to accelerate interactive performance of API calls and responses. This is controlled using the `Accept-Encoding` header on the request from the client and indicated by the `Content-Encoding` header in the server response. Unless the header is explicitly set, encoding defaults to disabled.

Table 3.3. Encoding Headers

Header Type	Name	Value
HTTP/1.1 Request	`Accept-Encoding`	`gzip`
HTTP/1.1 Response	`Content-Encoding`	`gzip`

3.5. Persistent Connections

By default, the API supports persistent connections via `HTTP/1.1` keepalives. All connections will be kept alive unless the connection header is set to close.

To prevent abuse, HTTP sessions have a timeout of 20 seconds before being closed.

Note

The server may close the connection at any time and clients should not rely on this behavior.

3.6. Paginated Collections

To reduce load on the service, list operations will return a maximum of 100 items at a time. To navigate the collection, the `limit` and `marker` parameters (for example, ? `limit=50&marker=1`) can be set in the URI. If a marker beyond the end of a list is given, an empty list is returned. Note that list operations never return 404 (itemNotFound) faults.

3.7. Efficient Polling with the `Changes-Since` Parameter

The ReST API allows you to poll for the status of certain operations by performing a **GET** on various URIs. Rather than re-downloading and re-parsing the full status at each polling interval, your ReST client may use the `Changes-Since` parameter to check for changes since a previous request. The `Changes-Since` time is specified as Unix time (the number of seconds since January 1, 1970, 00:00:00 UTC, not counting leap seconds). If nothing has changed since the `changes-since` time, a 304 (Not Modified) response is returned. If data has changed, only the items changed since the specified time is returned. For example, performing a **GET** against `https://URL/v1.0/1234/loadbalancers?changes-since=1244012982` would list all load balancers that have changed since Wed, 03 Jun 2009 07:09:42 UTC.

3.8. Limits

All accounts, by default, have a preconfigured set of thresholds (or limits) to manage capacity and prevent abuse of the system. The system recognizes two kinds of limits: rate limits and absolute limits. Rate limits are thresholds that are reset after a certain amount of time passes. Absolute limits are fixed.

Note

If the default limits are too low for your particular application, please contact Rackspace Cloud support to request an increase. All requests require reasonable justification.

3.8.1. Rate Limits

We specify rate limits in terms of both a human-readable wild-card URI and a machine-processable regular expression. The regular expression boundary matcher '^' takes effect after the root URI path. For example, the regular expression `^/v1.0/1234/loadbalancers` would match the bolded portion of the following URI: `https://ord.loadbalancers.api.rackspacecloud.com` **`/v1.0/1234/loadbalancers`**.

Table 3.4. Default Rate Limits

Verb	URI	RegEx	Default
GET	/v1.0/*	^/1.0/.*	5/second
GET	/v1.0/*	^/1.0/.*	100/minute
POST	/v1.0/*	^/1.0/.*	2/second
POST	/v1.0/*	^/1.0/.*	25/minute
PUT	/v1.0/*	^/1.0/.*	5/second
PUT	/v1.0/*	^/1.0/.*	50/minute
DELETE	/v1.0/*	^/1.0/.*	2/second
DELETE	/v1.0/*	^/1.0/.*	50/minute

Rate limits are applied in order relative to the verb, going from least to most specific. For example, although the threshold for **POST** to /v1.0/* is 25 per minute, one cannot **POST** to /v1.0/* more than 2 times per second because the rate limits for any **POST** is 2/second. In the event you exceed the thresholds established for your account, a 413 (Rate Control) HTTP response will be returned with a `Reply-After` header to notify the client when they can attempt to try again.

3.8.2. Absolute Limits

`Maximum number of load balancers` limits the total number of load balancers that may be associated with a given account.

Limit	Default
Maximum number of load balancers	20

3.8.3. Determining Limits Programmatically

Applications can programmatically determine current account limits using the `/limits` URI as follows:

Verb	URI	Description
GET	/limits	Returns the current limits for the account.

Normal Response Code(s): 200

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation does not require a request body.

Example 3.3. List Limits Response: XML

```
<limits xmlns="http://docs.openstack.org/common/api/v1.0">
    <rates>
        <rate uri="/v1.0/*" regex="^/1.0/.*">
            <limit
                verb="GET"
                value="600000"
                remaining="426852"
                unit="HOUR"
                next-available="2011-02-22T19:32:43.835Z"/>
        </rate>
    </rates>
</limits>
```

Example 3.4. List Limits Response: JSON

```
{
    "limits" : {
        "rate" : [
            {
                "uri" : "/v1.0/*",
                "regex" : "^/1.0/.*",
                "limit" : [
                    {
                        "verb" : "GET",
                        "value" : 600000,
                        "remaining" : 426852,
                        "unit" : "HOUR",
                        "next-available"
 : "2011-02-22T19:32:43.835Z"
                    }
                ]
            }
        ]
    }
}
```

4. API Operations

4.1. List Load Balancers

Verb	URI	Description	Representation
GET	/loadbalancers	List all load balancers configured for the account (IDs, names and status only)	XML, JSON, ATOM

Normal Response Code(s): 200

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation provides a list of all load balancers configured and associated with your account.

Load balancers which have been deleted will be shown in this list for at least 90 days after deletion. A deleted load balancer is immutable and irrecoverable. Only a limited set of attributes (`id`,`name`,`status`, `created`, and `updated`) will be returned in the response object.

This operation does not require a request body.

Example 4.1. List Load Balancers Response: XML

```
<?xml version="1.0" ?>
<loadBalancers xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
    <loadBalancer id="71" name="lb-site1" status="ACTIVE"
        protocol="HTTP" port="80" algorithm="RANDOM">
        <virtualIps>
            <virtualIp id="403" address="206.55.130.1" ipVersion="IPV4"
                type="PUBLIC" />
        </virtualIps>
        <created time="2010-12-13T15:38:27-06:00" />
        <updated time="2010-12-13T15:38:38-06:00" />
    </loadBalancer>
    <loadBalancer id="166" name="lb-site2" status="ACTIVE"
        protocol="HTTP" port="80" algorithm="RANDOM">
        <virtualIps>
            <virtualIp id="401" address="206.55.130.2" ipVersion="IPV4"
                type="PUBLIC" />
        </virtualIps>
        <created time="2010-12-13T15:38:27-06:00" />
        <updated time="2010-12-13T15:38:38-06:00" />
    </loadBalancer>
</loadBalancers>
```

Example 4.2. List Load Balancers Response: JSON


```
{
    "loadBalancers":[
        {
            "name":"lb-site1",
            "id":"71",
            "protocol":"HTTP",
            "port":"80",
            "algorithm":"RANDOM",
            "status":"ACTIVE",
            "virtualIps":[
                {
                    "id":"403",
                    "address":"206.55.130.1",
                    "type":"PUBLIC",
                    "ipVersion":"IPV4"
                }
            ],
            "created":{
                "time":"2010-11-30T03:23:42.000+0000"
            },
            "updated":{
                "time":"2010-11-30T03:23:44.000+0000"
            }
        },
        {
            "name":"lb-site2",
            "id":"166",
            "protocol":"HTTP",
            "port":"80",
            "algorithm":"RANDOM",
            "status":"ACTIVE",
            "virtualIps":[
                {
                    "id":"401",
                    "address":"206.55.130.2",
                    "type":"PUBLIC",
                    "ipVersion":"IPV4"
                }
            ],
            "created":{
                "time":"2010-11-30T03:23:42.000+0000"
            },
            "updated":{
                "time":"2010-11-30T03:23:44.000+0000"
            }
        }
    ]
}
```

4.2. List Load Balancer Details

Verb	URI	Description	Representations
GET	/loadbalancers/ loadBalancerId	List details of the specified load balancer	JSON, XML, ATOM

Normal Response Code(s): 200

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation provides detailed output for a specific load balancer configured and associated with your account. This operation is not capable of returning details for a load balancer which has been deleted.

This operation does not require a request body.

Example 4.3. List Load Balancer Details Request: XML

```
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
    id="2000"
    name="sample-loadbalancer"
    protocol="HTTP"
    port="80"
    algorithm="RANDOM"
    status="ACTIVE">
    <connectionLogging enabled="false" />
    <virtualIps>
        <virtualIp
            id="1000"
            address="206.10.10.210"
            type="PUBLIC"
            ipVersion="IPV4" />
    </virtualIps>
    <nodes>
        <node
            nodeId="1041"
            address="10.1.1.1"
            port="80"
            condition="ENABLED"
            status="ONLINE" />
        <node
            nodeId="1411"
            address="10.1.1.2"
            port="80"
            condition="ENABLED"
            status="ONLINE" />
    </nodes>
    <sessionPersistence persistenceType="HTTP_COOKIE"/>
    <connectionThrottle
        minConnections="10"
        maxConnections="100"
        maxConnectionRate="50"
        rateInterval="60" />
    <cluster name="c1.dfw1" />
    <created time="2010-11-30T03:23:42Z" />
    <updated time="2010-11-30T03:23:44Z" />
</loadBalancer>
```

Example 4.4. List Load Balancers Details Response: JSON


```
{
    "loadBalancers":[
        {
            "name":"lb-site1",
            "id":"71",
            "protocol":"HTTP",
            "port":"80",
            "algorithm":"RANDOM",
            "status":"ACTIVE",
            "virtualIps":[
                {
                    "id":"403",
                    "address":"206.55.130.1",
                    "type":"PUBLIC",
                    "ipVersion":"IPV4"
                }
            ],
            "created":{
                "time":"2010-11-30T03:23:42.000+0000"
            },
            "updated":{
                "time":"2010-11-30T03:23:44.000+0000"
            }
        },
        {
            "name":"lb-site2",
            "id":"166",
            "protocol":"HTTP",
            "port":"80",
            "algorithm":"RANDOM",
            "status":"ACTIVE",
            "virtualIps":[
                {
                    "id":"401",
                    "address":"206.55.130.2",
                    "type":"PUBLIC",
                    "ipVersion":"IPV4"
                }
            ],
            "created":{
                "time":"2010-11-30T03:23:42.000+0000"
            },
            "updated":{
                "time":"2010-11-30T03:23:44.000+0000"
            }
        }
    ]
}
```

4.3. Create Load Balancer

Verb	URI	Description
POST	/loadbalancers	Create a new load balancer with the configuration defined by the request.

Normal Response Code(s): 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation asynchronously provisions a new load balancer based on the configuration defined in the request object. Once the request is validated and progress has started on the provisioning process, a response object will be returned. The object will contain a unique identifier and status of the request. Using the identifier, the caller can check on the progress of the operation by performing a **GET** on `loadbalancers/id`. If the corresponding request cannot be fulfilled due to insufficient or invalid data, an HTTP 400 (Bad Request) error response will be returned with information regarding the nature of the failure in the body of the response. Failures in the validation process are non-recoverable and require the caller to correct the cause of the failure and **POST** the request again.

Note

A load balancer's name must be less than or equal to 128 characters.

Note

Users may configure all documented features of the load balancer at creation time by simply providing the additional elements / attributes in the request. Refer to the subsequent sections of this specification for an overview of all features the load balancing service supports.

Example 4.5. Create Load Balancer (Required Attributes) Request: XML

```
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    name="a-new-loadbalancer"
    port="80"
    protocol="HTTP">
    <virtualIps>
        <virtualIp type="PUBLIC"/>
    </virtualIps>
    <nodes>
        <node address="10.1.1.1" port="80" condition="ENABLED"/>
    </nodes>
</loadBalancer>
```

Example 4.6. Create Load Balancer (Required Attributes) Request: JSON

```
{
    "loadBalancer": {
        "name": "a-new-loadbalancer",
        "port": "80",
        "protocol": "HTTP",
        "virtualIps": [
            {
                "type": "PUBLIC"
            }
        ],
        "nodes": [
            {
                "address": "10.1.1.1",
                "port": "80",
                "condition": "ENABLED"
            }
        ]
    }
}
```

In order to conserve IPv4 address space, Rackspace highly recommends sharing Virtual IPs between your load balancers. If you have at least one load balancer, you may create subsequent load balancers that share a single virtual IP by issuing a **POST** and supplying a virtual Ip ID instead of a type. Additionally, this feature is highly desirable if you wish to load balance both an unsecured and secure protocol using one IP / DNS name (for example, HTTP and HTTPS).

Note

Load balancers sharing a virtual IP *must* utilize a unique port.

Example 4.7. Create Load Balancer (Required Attributes with Shared IP) Request: XML

```
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    name="a-new-loadbalancer"
    port="80"
    protocol="HTTP">
    <virtualIps>
        <virtualIp id="2341"/>
    </virtualIps>
    <nodes>
        <node address="10.1.1.1" port="80" condition="ENABLED" />
    </nodes>
</loadBalancer>
```

Example 4.8. Create Load Balancer (Required Attributes with Shared IP) Request: JSON

```
{
    "loadBalancer":{
        "name":"a-new-loadbalancer",
        "port":"80",
        "protocol":"HTTP",
        "virtualIps":[
            {
                "id":"2341"
            }
        ],
        "nodes":[
            {
                "address":"10.1.1.1",
                "port":"80",
                "condition":"ENABLED"
            }
        ]
    }
}
```

Example 4.9. Create Load Balancer (Required Attributes with Shared IP) Response: XML

```
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    id="144"
    name="a-new-loadbalancer"
    algorithm="RANDOM"
    protocol="HTTP"
    port="83"
    status="BUILD">
    <virtualIps>
        <virtualIp
            id="39"
            address="206.10.10.210"
            ipVersion="IPV4"
            type="PUBLIC" />
    </virtualIps>
    <nodes>
        <node
            id="653"
            address="10.1.1.1"
            port="80"
            condition="ENABLED"
            status="ONLINE"
            weight="1" />
    </nodes>
    <cluster name="ztm-n03.staging1.lbaas.rackspace.net" />
    <created time="2011-02-08T21:19:55Z" />
    <updated time="2011-02-08T21:19:55Z" />
    <connectionLogging enabled="false" />
</loadBalancer>
```

4.4. Remove Load Balancer

Verb	URI	Description
DELETE	/loadbalancers/ loadBalancerId	Removes a load balancer from your account

Normal Response Code(s): 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

The remove load balancer function removes the specified load balancer and its associated configuration from the account. Any and all configuration data is immediately purged and is not recoverable.

This operation does not require a request body.

This operation does not return a response body.

4.5. List, Add, Modify, and Remove Nodes

Verb	URI	Description
GET	/loadbalancers/ loadBalancerId/nodes	List node(s) configured for the load balancer
POST	/loadbalancers/ loadBalancerId/nodes	Add a new node to the load balancer
PUT	/loadbalancers/ loadBalancerId/nodes/ nodeId	Modifies the configuration of a node on the load balancer
DELETE	/loadbalancers/ loadBalancerId/nodes/ nodeId	Removes a node from the load balancer
GET	/loadbalancers/ loadBalancerId/nodes/ nodeId	List details for a specific node

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

The nodes defined by the load balancer are responsible for servicing the requests received through the load balancer's virtual IP. By default, the load balancer employs a basic health check that ensures the node is listening on its defined port. The node is checked at the time of addition and at regular intervals as defined by the load balancer health check configuration. If a back-end node is not listening on its port or does not meet the conditions of the defined

active health check for the load balancer, then the load balancer will not forward connections and its status will be listed as`OFFLINE`. Only nodes that are in an `ONLINE` status will receive and be able to service traffic from the load balancer.

All nodes have an associated status that indicates whether the node is online, offline, or draining. Only nodes that are in an `ONLINE` status will receive and be able to service traffic from the load balancer. The `OFFLINE` status represents a node that cannot accept or service traffic. A node in `DRAINING` status represents a node that stops the traffic manager from sending any additional new connections to the node, but honors established sessions. If the traffic manager receives a request and session persistence requires that the node is used, the traffic manager will use it. The status is determined by the passive or active health monitors.

If the `WEIGHTED_ROUND_ROBIN` load balancer algorithm mode is selected, then the caller should assign the relevant weights to the node as part of the weight attribute of the node element. When the algorithm of the load balancer is changed to `WEIGHTED_ROUND_ROBIN` and the nodes do not already have an assigned weight, the service will automatically set the weight to "1" for all nodes.

When a node is added, it is assigned a unique identifier that can be used for mutating operations such as changing the port and condition or removing it. Every load balancer is dual-homed on both the public Internet and ServiceNet; as a result, nodes can either be internal ServiceNet addresses or addresses on the public Internet.

This operation does not require a request body.

Example 4.10. List Node Response: XML

```
<node
    id="410"
    address="10.1.1.1"
    port="80"
    condition="ENABLED"
    status="ONLINE"/>
```

Example 4.11. Add Node Request: XML

```
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
    <node
        address="10.1.1.1"
        port="80"
        condition="ENABLED"/>
</nodes>
```

Example 4.12. Add Node Response: XML

```
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
    <node
        address="10.1.1.1"
        port="80"
        condition="ENABLED"
        status="ONLINE" />
</nodes>
```

Example 4.13. Add Node Request: JSON

```
{"nodes": [
    {
        "port": 80,
        "condition": "ENABLED",
        "address": "10.1.1.1"
    }
  ]
}
```

Example 4.14. Add Node Response: JSON

```
{"nodes": [
    {
        "address": "10.1.1.1",
        "port": "80",
        "condition": "ENABLED",
        "status": "PENDING_UPDATE"
    }
  ]
}
```

Note

The node's IP and Port are immutable attributes and cannot be modified by the caller for a **PUT** request. Supplying an unsupported attribute will result in a 400 (badRequest) fault. Additionally, a load balancer supports a maximum of 25 nodes.

Example 4.15. Update Node Attributes Request: XML

```
<node xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
    condition="DISABLED"/>
```

Example 4.16. Update Node Attributes Request: JSON

```
{
    "node":{
    "condition":"DISABLED"
    }
}
```

4.6. Update Load Balancer Attributes

Verb	URI	Description
PUT	/loadbalancers/ loadBalancerId	Update the properties of a load balancer

Normal Response Code(s): 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation asynchronously updates the attributes of the specified load balancer. Upon successful validation of the request, the service will return a 202 (Accepted) response code. A caller can subscribe to event notifications through the notification service or poll the load balancer with its ID to wait for the changes to be applied and the load balancer to return to an ACTIVE status.

This operation allows the caller to change one or more of the following attributes:

- `name`
- `algorithm`
- `protocol`
- `port`

This operation does not return a response body.

Note

The load balancer's ID and status are immutable attributes and cannot be modified by the caller. Supplying an unsupported attribute will result in a 400 (badRequest) fault.

Example 4.17. Update Load Balancer Attributes Request: XML

```
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    name="sample-loadbalancer"
    algorithm="RANDOM"
    protocol="HTTP"
    port="80" />
```

Example 4.18. Update Load Balancer Attributes Request: JSON

```
{
    "name": "sample-loadbalancer",
    "algorithm": "RANDOM",
    "protocol": "HTTP",
    "port": "80",
    "connectionLogging": "true"
}
```

4.7. Virtual IP Management

Verb	URI	Description
GET	/loadbalancers/ loadBalancerId/virtualips	List all associated virtual IPs
DELETE	/loadbalancers/ loadBalancerId/virtualIpId	Remove the Virtual IP

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

A virtual IP (VIP) makes a load balancer accessible by clients. The load balancing service supports either a public VIP, routable on the public Internet, or a ServiceNet address, routable only within the region in which the load balancer resides.

This request does not require a request body.

Note

All load balancers must have at least one virtual IP associated with them at all times. Attempting to delete the last virtual IP will result in a 400 (badRequest) fault.

Table 4.1. Virtual IP Types

Name	Description
PUBLIC	An address that is routable on the public Internet
SERVICENET	An address that is routable only on ServiceNet

Example 4.19. List Virtual IPs Response: XML

```
<virtualIps xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0">
    <virtualIp
        id="1000"
        address="206.10.10.210"
        type="PUBLIC"/>
</virtualIps>
```

Example 4.20. List Virtual IPs Response: JSON

```
{"virtualIps": [
    {
    "id": "1000",
    "address": "206.10.10.210",
    "type": "PUBLIC"
    }
  ]
}
```

4.8. Usage Reporting

Name	URI	Description
GET	/loadbalancers/loadBalancerId/usage	List current and historical usage
GET	/loadbalancers/loadBalancerId/usage/current	List current usage

Normal Response Code(s): 200

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

The load balancer usage reports provide a view of all transfer, average number of connections and number of virtual IPs associated with the load balancing service. Current usage represents all usage recorded within the last 24 hours. Values for both `incomingTransfer` and `outgoingTransfer` are expressed in bytes transferred. The optional `startTime` and `endTime` parameters can be used to filter all usage. If the `startTime` parameter is supplied, but the `endTime` parameter is not then all usage beginning with the `startTime` will be provided. Likewise, if the `endTime` parameter is supplied but the `startTime` parameter is not then all usage will be returned up to the `endTime` specified.

Note

Usage is available for up to 90 days of service activity.

This operation does not require a request body.

Example 4.21. Report Load Balancer Usage Response: XML

```
<loadBalancerUsage xmlns="http://docs.openstack.org/loadbalancers/
api/v1.0">
    <loadBalancerUsageRecord
        id="394"
        averageNumConnections="0.0"
        incomingTransfer="0"
        outgoingTransfer="0"
        numVips="1"
        numPolls="32"
        startTime="2010-12-21T12:32:07-06:00"
        endTime="2010-12-21T16:23:54-06:00" />
    <loadBalancerUsageRecord
        id="473"
        averageNumConnections="0.0"
        incomingTransfer="0"
        outgoingTransfer="0"
        numVips="2"
        numPolls="5"
        startTime="2010-12-21T12:32:07-06:00"
        endTime="2010-12-21T12:36:30-06:00" />
</loadBalancerUsage>
```

Example 4.22. Report Load Balancer Usage Response: JSON

```
{
    "loadBalancerUsageRecords": [
    {
    "id": "394",
    "averageNumConnections": "0.0",
    "incomingTransfer": "0",
    "outgoingTransfer": "0",
    "numVips": "1",
    "numPolls" "32",
    "startTime": "2010-12-21T12:32:07-06:00",
    "endTime": "2010-12-21T16:23:54-06:00"
    },
    {
    "id": "473",
    "averageNumConnections": "0.0",
    "incomingTransfer": "0",
    "outgoingTransfer": "0",
    "numVips": "2",
    "numPolls": "5",
    "startTime": "2010-12-21T12:32:07-06:00",
    "endTime": "2010-12-21T12:36:30-06:00"
    }
  ]
}
```

Example 4.23. Report Account Billing Response: XML

```
<accountBilling xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0" accountId="1106">
    <accountUsage>
        <accountUsageRecord numLoadBalancers="2" numPublicVips="1"
            numServicenetVips="0" startTime="2010-12-01T14:09:14-06:00"/>
    </accountUsage>
    <loadBalancerUsage loadBalancerId="66" loadBalancerName="My first
 loadbalancer">
        <loadBalancerUsageRecord id="394" averageNumConnections="0.0"
            incomingTransfer="0" outgoingTransfer="0" numVips="1"
            numPolls="32" startTime="2010-12-21T12:32:07-06:00"
            endTime="2010-12-21T16:23:54-06:00"/>
        <loadBalancerUsageRecord id="473" averageNumConnections="0.0"
            incomingTransfer="0" outgoingTransfer="0" numVips="2"
            numPolls="5" startTime="2010-12-21T12:32:07-06:00"
            endTime="2010-12-21T12:36:30-06:00"/>
        <loadBalancerUsageRecord id="474" averageNumConnections="0.0"
            incomingTransfer="0" outgoingTransfer="0" numVips="2"
            numPolls="5" startTime="2010-12-21T12:32:07-06:00"
            endTime="2010-12-21T12:36:30-06:00"/>
        <loadBalancerUsageRecord id="475" averageNumConnections="0.0"
            incomingTransfer="0" outgoingTransfer="0" numVips="2"
            numPolls="5" startTime="2010-12-21T12:32:07-06:00"
            endTime="2010-12-21T12:36:30-06:00"/>
    </loadBalancerUsage>
    <loadBalancerUsage loadBalancerId="77" loadBalancerName="My second
 loadbalancer">
        <loadBalancerUsageRecord id="394" averageNumConnections="0.0"
            incomingTransfer="0" outgoingTransfer="0" numVips="1"
            numPolls="32" startTime="2010-12-21T12:32:07-06:00"
            endTime="2010-12-21T16:23:54-06:00"/>
        <loadBalancerUsageRecord id="473" averageNumConnections="0.0"

  incomingTransfer="0" outgoingTransfer="0" numVips="2" numPolls="5"
            startTime="2010-12-21T12:32:07-06:00"
            endTime="2010-12-21T12:36:30-06:00"/>
    </loadBalancerUsage>
</accountBilling>
```

4.9. Access List Management

Name	URI	Description
GET	/loadbalancers/loadBalancerId/accesslist	List the access list
POST	/loadbalancers/loadBalancerId/accesslist	Create a new access list or append to an existing access list
DELETE	/loadbalancers/loadBalancerId/accesslist/networkItemId	Remove a network item from the access list
DELETE	/loadbalancers/loadBalancerId/accesslist	Remove the entire access list

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

The access list management feature allows for fine-grained network access controls to be applied to the load balancer's virtual IP address. A single IP address, multiple IP addresses, or entire network subnets can be added as a networkItem. Items that are configured with the `ALLOW` type will always take precedence over items with the `DENY` type. To reject traffic from all items, except for those with the `ALLOW` type, a networkItem with an address of "0.0.0.0/0" should be added with a `DENY` type.

When issuing a **POST** to add to an access list, one or more network items are required. If a populated access list already exists for the load balancer, it will be appended with subsequent **POST** requests. A single address or subnet definition is considered unique and cannot be duplicated between items in an access list. There are two **DELETE** operations for the access list. One allows for deletion of the entire access list and the other for deletion of a specific network item in the access list.

This operation does not require a request body.

Example 4.24. Retrieve Access List Response: XML

```
<accessList xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0">
    <networkItem
        id="1000"
        address="206.160.165.40"
        type="ALLOW" />
    <networkItem
        id="1001"
        address="206.160.165.0/24"
        type="DENY" />
</accessList>
```

Example 4.25. Retrieve Access List Response: JSON


```
{
    "networkItems": [
        {
            "address": "206.160.163.21",
            "id": 23,
            "type": "DENY"
        },
        {
            "address": "206.160.165.11",
            "id": 24,
            "type": "DENY"
        },
        {
            "address": "206.160.163.21",
            "id": 25,
            "type": "DENY"
        },
        {
            "address": "206.160.165.11",
            "id": 26,
            "type": "DENY"
        },
        {
            "address": "206.160.123.11",
            "id": 27,
            "type": "DENY"
        },
        {
            "address": "206.160.122.21",
            "id": 28,
            "type": "DENY"
        },
        {
            "address": "206.140.123.11",
            "id": 29,
            "type": "DENY"
        },
        {
            "address": "206.140.122.21",
            "id": 30,
            "type": "DENY"
        }
    ]
}
```

Example 4.26. Update Access List Attributes Request: XML

```
<accessList xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0">
    <networkItem
        address="206.160.165.1"
        type="ALLOW" />
    <networkItem
        address="206.160.165.2"
        type="DENY" />
</accessList>
```

Example 4.27. Update Access List Attributes Request: JSON

```
{
    "AccessList": [
        {
            "address": "206.160.163.21",
            "type": "DENY"
        },
        {
            "address": "206.160.165.11",
            "type": "DENY"
        }
    ]
}
```

4.10. Active Health Monitoring

Verb	URI	Description
GET	/loadbalancers/ loadBalancerId/ healthmonitor	Retrieves the health monitor configuration, if one exists
PUT	/loadbalancers/ loadBalancerId/ healthmonitor	Updates the settings for a health monitor
DELETE	/loadbalancers/ loadBalancerId/ healthmonitor	Removes the health monitor

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

The load balancing service includes a health monitoring operation which periodically checks your back-end nodes to ensure they are responding correctly. If a node is not responding, it is removed from rotation until the health monitor determines that the node is functional. In addition to being performed periodically, the health check also is performed against every node that is added to ensure that the node is operating properly before allowing it to service traffic. Only one health monitor is allowed to be enabled on a load balancer at a time.

Every health monitor has a `type` attribute to signify what kind of monitor it is.

Table 4.2. Health Monitor Types

Name	Description
CONNECT	Health monitor is a connect monitor
HTTP	Health monitor is a HTTP monitor
HTTPS	Health monitor is a HTTPS monitor

Example 4.28. Monitor Connections Response: XML

```
<healthMonitor xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    type="CONNECT"
    delay="10"
    timeout="10"
    attemptsBeforeDeactivation="3" />
```

Example 4.29. Monitor Connections Response: JSON

```
{
    "type": "CONNECT",
    "delay": "10",
    "timeout": "10",
    "attemptsBeforeDeactivation": "3"
}
```

Example 4.30. Monitor HTTP Response: XML

```
<healthMonitor xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    type="HTTP"
    delay="10"
    timeout="10"
    attemptsBeforeDeactivation="3"
    path="/"
    statusRegex="^[234][0-9][0-9]$"
    bodyRegex="" />
```

Example 4.31. Monitor HTTPs Response: XML

```
<healthMonitor xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    type="HTTPS"
    delay="10"
    timeout="10"
    attemptsBeforeDeactivation="3"
    path="/"
    statusRegex="^[234][0-9][0-9]$"
    bodyRegex="" />
```

Note

Attributes that are required for a **POST** request are displayed in the samples, unless otherwise specified.

Example 4.32. Monitor Connections Request: XML

```
<healthMonitor xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0"
    type="CONNECT"
    delay="10"
    timeout="10"
    attemptsBeforeDeactivation="3" />
```

Example 4.33. Monitor Connections Request: JSON

```
{
    "type": "CONNECT",
    "delay": "10",
    "timeout": "10",
    "attemptsBeforeDeactivation": "3"
}
```

4.10.1. Connection Monitor

The monitor connects to each node on its defined port to ensure that the service is listening properly. The connect monitor is the most basic type of health check and does no post-processing or protocol specific health checks. It includes several configurable properties:

- `delay`: The minimum number of seconds to wait before executing the health monitor.
- `timeout`: Maximum number of seconds to wait for a connection to be established before timing out.
- `attemptsBeforeDeactivation`: Number of permissible monitor failures before removing a node from rotation.

4.10.2. HTTP/HTTPS Monitor

The HTTP & HTTPS monitor is a more intelligent monitor that is capable of processing a HTTP or HTTPS response to determine the condition of a node. It supports the same basic properties as the connect monitor and includes three additional attributes that are used to evaluate the HTTP response.

- `delay`: The minimum number of seconds to wait before executing the health monitor.
- `timeout`: Maximum number of seconds to wait for a connection to be established before timing out.
- `attemptsBeforeDeactivation`: Number of permissible monitor failures before removing a node from rotation.
- `path`: The HTTP path that will be used in the sample request
- `statusRegex`: A regular expression that will be used to evaluate the HTTP status code returned in the response.
- `bodyRegex`: A regular expression that will be used to evaluate the contents of the body of the response.

4.11. Session Persistence

Verb	URI	Description
GET	/loadbalancers/ loadBalancerId/ sessionpersistence	List session persistence configuration
PUT	/loadbalancers/ loadBalancerId/ sessionpersistence	Enable session persistence
DELETE	/loadbalancers/ loadBalancerId/ sessionpersistence	Disable session persistence

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

Session persistence is a feature of the load balancing service which forces subsequent requests from clients to be directed to the same node. This is common with many web applications that do not inherently share application state between back-end servers.

Table 4.3. Session Persistence Modes

Name	Description
HTTP_COOKIE	A session persistence mechanism that inserts an HTTP cookie and is used to determine the destination back-end node. This is supported for HTTP load balancing only.

Example 4.34. List Session Persistence Configuration Response: XML

```
<sessionPersistence xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0" persistenceType="HTTP_COOKIE"/>
```

Example 4.35. List Session Persistence Configuration Response: JSON

```
{
   "sessionPersistence":{
      "persistenceType":"HTTP_COOKIE"
   }
}
```

Example 4.36. Set Session Persistence Type Request: XML

```
<sessionPersistence xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0" persistenceType="HTTP_COOKIE"/>
```

Example 4.37. Set Session Persistence Type Request: JSON

```
{
   "sessionPersistence":{
      "persistenceType":"HTTP_COOKIE"
   }
}
```

4.12. Connection Logging

Verb	URI	Description
GET	/loadbalancers/ loadBalancerId/ connectionlogging	View current configuration of connection logging
PUT	/loadbalancers/ loadBalancerId/ connectionlogging	Enable or disable connection logging

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation allows the user to enable/disable connection logging.

This operation does not require a request body.

Example 4.38. List Connection Logging Configuration Response: XML

```
<connectionLogging xmlns="http://docs.openstack.org/loadbalancers/
api/v1.0" enabled="true"/>
```

Example 4.39. List Connection Logging Configuration Response: JSON

```
{
   "connectionLogging": {
      "enabled": "true"
   }
}
```

Example 4.40. Enable Connection Logging Request: XML

```
<connectionlogging xmlns="http://docs.openstack.org/loadbalancers/
api/v1.0" enabled="true"/>
```

Example 4.41. Enable Connection Logging Request: JSON

```
{
   "connectionlogging":{
      "enabled":"true"
   }
}
```

4.13. Connection Throttling

Verb	URI	Description
GET	/loadbalancers/ loadBalancerId/ connectionthrottle	List connection throttling configuration
PUT	/loadbalancers/ loadBalancerId/ connectionthrottle	Update throttling configuration
DELETE	/loadbalancers/ loadBalancerId/ connectionthrottle	Remove connection throttling configurations

Normal Response Code(s): 200, 202

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

The connection throttling feature imposes limits on the number of connections per IP address to help mitigate malicious or abusive traffic to your applications. The following properties can be configured based on the traffic patterns for your sites.

- `minConnection`: Allow at least this number of connections per IP address before applying throttling restrictions.
- `maxConnections`: Maximum number of connection to allow for a single IP address.
- `macConnectionRate`: Maximum number of connections allowed from a single IP address in the defined `rateInterval`.
- `rateInterval`: Frequency (in seconds) at which the `maxConnectionRate` is assessed. For example, a `maxConnectionRate` of 30 with a `rateInterval` of 60 would allow a maximum of 30 connections per minute for a single IP address.

This operation does not require a request body.

Example 4.42. List Connection Throttling Configuration Response: XML

```
<connectionThrottle xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0"
    minConnections="10"
    maxConnections="100"
    maxConnectionRate="50"
    rateInterval="60" />
```

Example 4.43. List Connection Throttling Configuration Response: JSON

```
{
    "maxConnections": 100,
    "minConnections": 10,
    "maxConnectionRate": 50,
    "rateInterval": 60
}
```

Example 4.44. Update Connection Throttling Configuration Request: XML

```
<connectionThrottle xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0"
    minConnections="10"
    maxConnections="100"
    maxConnectionRate="50"
    rateInterval="60" />
```

Example 4.45. Update Connection Throttling Configuration Request: JSON

```
{
    "maxConnections": 10,
    "minConnections": 100,
    "maxConnectionRate": 50,
    "rateInterval": 60
}
```

4.14. Load Balancing Protocols

Verb	URI	Description
GET	/loadbalancers/protocols	List all supported load balancing protocols

Normal Response Code(s): 200

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

All load balancers must define the protocol of the service which is being load balanced. The protocol selection should be based on the protocol of the back-end nodes. When configuring a load balancer, the port selected will be the default port for the given protocol unless otherwise specified.

This operation does not require a request body.

Example 4.46. List Load Balancing Protocols Response: XML

```
<protocols xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0">
    <protocol name="FTP" port="21" />
    <protocol name="HTTP" port="80" />
    <protocol name="IMAPv4" port="143" />
    <protocol name="POP3" port="110" />
    <protocol name="LDAP" port="389" />
    <protocol name="LDAPS" port="636" />
    <protocol name="HTTPS" port="443" />
    <protocol name="IMAPS" port="993" />
    <protocol name="POP3S" port="995" />
    <protocol name="SMTP" port="25" />
</protocols>
```

Example 4.47. List Load Balancing Protocols Response: JSON

```
{"protocols": [
        {
            "name": "HTTP",
            "port": "80"
        },
        {
            "name": "FTP",
            "port": "21"
        },
        {
            "name": "IMAPv4",
            "port": "143"
        },
        {
            "name": "POP3",
            "port": "110"
        },
        {
            "name": "SMTP",
            "port": "25"
        },
        {
            "name": "LDAP",
            "port": "389"
        },
        {
            "name": "HTTPS",
            "port": "443"
        },
        {
            "name": "IMAPS",
            "port": "993"
        },
        {
            "name": "POP3S",
            "port": "995"
        },
        {
            "name": "LDAPS",
            "port": "636"
        }
    ]
}
```

4.15. Load Balancing Algorithms

Verb	URI	Description
GET	/loadbalancers/algorithms	List all supported load balancing algorithms

Normal Response Code(s): 200

Error Response Code(s): loadbalancerFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

All load balancers utilize an algorithm that defines how traffic should be directed between back-end nodes. The default algorithm for newly created load balancers is `RANDOM`, which can be overridden at creation time or changed after the load balancer has been initially provisioned. The algorithm name is to be constant within a major revision of the load balancing API, though new algorithms may be created with a unique algorithm name within a given major revision of the service API.

This operation does not require a request body.

Table 4.4. Load Balancing Algorithms

Name	Description
`LEAST_CONNECTIONS`	The node with the lowest number of connections will receive requests
`RANDOM`	Back-end servers are selected at random
`ROUND_ROBIN`	Connections are routed to each of the back-end servers in turn
`WEIGHTED_LEAST_CONNECTIONS`	Each request will be assigned to a node based on the number of concurrent connections to the node and its weight
`WEIGHTED_ROUND_ROBIN`	A round robin algorithm, but with different proportions of traffic being directed to the back-end nodes. Weights must be defined as part of the load balancer's node configuration.

Example 4.48. List Load Balancing Algorithms Response: XML

```
<algorithms xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0">
    <algorithm name="LEAST_CONNECTIONS" />
    <algorithm name="RANDOM" />
    <algorithm name="ROUND_ROBIN" />
    <algorithm name="WEIGHTED_LEAST_CONNECTIONS" />
    <algorithm name="WEIGHTED_ROUND_ROBIN" />
</algorithms>
```

Example 4.49. List Load Balancing Algorithms Response: JSON

```
{"algorithms": [
        {
            "name": "LEAST_CONNECTIONS"
        },
        {
            "name": "RANDOM"
        },
        {
            "name": "ROUND_ROBIN"
        },
        {
            "name": "WEIGHTED_LEAST_CONNECTIONS"
        },
        {
            "name": "WEIGHTED_ROUND_ROBIN"
        }
    ]
}
```

4.16. Load Balancer Status

All load balancers have a status attribute to signify the current configuration status of the device. This status is immutable by the caller and is updated automatically based on state changes within the service. When a load balancer is first created, it will be placed into a build status where the configuration is being generated and applied based on the request. Once the configuration is applied and finalized, it will be in an ACTIVE status. In the event of a configuration change or update, the status of the load balancer will change to PENDING_UPDATE to signify configuration changes are in progress but have not yet been finalized. Load balancers in a SUSPENDED status are configured to reject traffic and will not forward requests to back-end nodes.

Table 4.5. Load Balancer Statuses

Name	Description
ACTIVE	Load balancer is configured properly and ready to serve traffic to incoming requests via the configured virtual IPs.
BUILD	Load balancer is being provisioned for the first time and configuration is being applied to bring the service online. The service will not yet be ready to serve incoming requests.
PENDING_UPDATE	Load balancer is online, but configuration changes are being applied to update the service based on a previous request.
PENDING_DELETE	Load balancer is online, but configuration changes are being applied to begin deletion of the service based on a previous request.
SUSPENDED	Load balancer has been taken offline and disabled; contact support.
ERROR	The system encountered an error when attempting to configure the laod balancer; contact support

4.17. Node Condition

Every node in the load balancer has an associated condition which determines its role within the load balancer.

Table 4.6. Load Balancer Node Conditions

Name	Description
ENABLED	Node is permitted to accept new connections
DISABLED	Node is not permitted to accept any new connections regardless of session persistence configuration. Existing connections are forcibly terminated
DRAINING	Node is allowed to service existing established connections and connections that are being directed to it as a result of the session persistence configuration

5. API Faults

API calls that return an error will return one of the following fault objects. All fault objects will extend from the base fault, `serviceFault`, for easier exception handling for languages that support it.

5.1. serviceFault

The `serviceFault` and by extension all other faults include `message` and `detail` elements which contain strings describing the nature of the fault as well as a `code` attribute representing the HTTP response code for convenience. The `code` attribute of the fault is for the convenience of the caller so that they may retrieve the response code from the HTTP response headers or directly from the fault object if they choose. The caller should not expect the `serviceFault` to be returned directly but should instead expect only one of the child faults to be returned.

5.2. loadBalancerFault

The `loadBalancerFault` fault shall be returned in the event that an error occurred during a loadbalancer operation.

```
<loadBalancerFault code="401" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>Invalid authentication token. Please renew</message>
</loadBalancerFault>
```

5.3. badRequest

This fault indicates that the data in the request object is invalid; for example, a string was used in a parameter that was expecting an integer. The fault will wrap validation errors.

```
<badRequest xmlns="http://docs.openstack.org/loadbalancers/api/
v1.0" code="400">
    <message>Validation fault</message>
    <details>The object is not valid</details>
        <validationErrors>
            <message>Node ip is invalid. Please specify a valid
 ip.</message>
        </validationErrors>
</badRequest>
```

5.4. itemNotFound

```
<itemNotFound code="404" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>Object not Found</message>
</itemNotFound>
```

5.5. overLimit

This fault is returned when the user has exceeded a currently allocated limit.

```
<overLimit code="413" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>Your account is currently over the limit so your
 request could not be processed.</message>
</overLimit>
```

5.6. unauthorized

This fault is returned when the user is not authorized to perform an attempted operation.

```
<unauthorized code="404" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>You are not authorized to execute this operation.</
message>
</unauthorized>
```

5.7. outOfVirtualIps

This fault indicates that there are no virtual IPs left to assign to a new loadbalancer. In practice, this fault should not occur, as virtual IPs will be ordered as capacity is required. If you do experience this fault, contact support so that we may make more IPs available.

```
<outOfVirtualIps code="500" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>
        Out of virtual IPs. Please contact support so they can
 allocate more virtual IPs.
    </message>
</outOfVirtualips>
```

5.8. immutableEntity

This fault is returned when a user attempts to modify an item that is not currently in a state that allows modification. For example, load balancers in a status of PENDING_UPDATE,BUILD, or DELETED may not be modified.

```
<immutableEntity code="422" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>The object at the specified URI is immutable and can
 not be overwritten.</message>
</immutableEntity>
```

5.9. unprocessableEntity

This fault is returned when an operation is requested on an item that does not support the operation, but the request is properly formed.

```
<unprocessableEntity code="422" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>The Object at the specified URI is unprocessable.</
message>
</unprocessableEntity>
```

5.10. serviceUnavailable

This fault is returned when the service is unavailable, such as when the service is undergoing maintenance. Note that this does not necessarily mean that the currently configured loadbalancers are unable to process traffic; it simply means that the API is currently unable to service requests.

```
<serviceUnavailable code="500" xmlns="http://docs.openstack.org/
loadbalancers/api/v1.0">
    <message>The Load balancing service is currently not
 available</message>
</serviceUnavailable>
```