Cloud Load Balancers Developer Guide

Cloud Load Balancers Developer Guide

API v1.1

Publication date 2011-04-19

Abstract

This document is intended for software developers interested in developing applications using the Cloud Load Balancers Application Program Interface (API).

Table of Contents

List of Tables

List of Examples

Chapter 1. Overview

1. Intended Audience

This guide is intended for software developers who want to create applications using the 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

2. Document Change History

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

Revision Date	Summary of Changes
Apr. 19, 2011	• Added details to support initial GA release.
Mar. 2, 2011	• Revised code samples and formatting to address initial beta feedback.
Feb. 23, 2011	• Initial release for public beta.

Chapter 2. Concepts

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

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. Virtual IP

A virtual IP is an Internet Protocol (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.

3. Node

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

4. Health Monitor

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

4.1. Passive Health Monitor

By default, all load balancing configurations utilize a passive health monitor, which is the default monitoring and does not require configuration from the user. If the passive health monitoring determines that a node is down, unreachable or malfunctioning, it puts the node in an OFFLINE state and stops sending traffic to it.

4.2. Active Health Monitor

Active health monitoring is a technique that uses synthetic transactions executed at periodic intervals to determine the condition of a node. When active monitoring is enabled, it takes over the monitoring of the nodes, and passive monitoring is disabled. Conversely, when active monitoring configuration is removed by the user, passive monitoring is re-enabled for the nodes of the load balancer

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.

5. Session Persistence

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

6. Connection Logging

The connection logging feature allows for retrieving access logs (for HTTP-based protocol traffic) or connection and transfer logs (for all other traffic)

Chapter 3. General API Information

Ideas explained here are relevant to all operations of the OpenStack API.

1. Authentication

Common to OpenStack API

2. Service Access/Endpoints

Common to OpenStack API

3. Request/Response Types

The 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.

Table 3.1. JSON and XML Response Formats

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

4. Content Compression

Common to all of OpenStackAPI.

5. Persistent Connections

Common to all of OpenStackAPI.

6. Paginated Collections

Common to all of OpenStackAPI.

7. Limits

Limits that are queriable beforehand on things like max name length, max VIPs per load balancer, max health monitors per load balancer, etc

7.1. Determining Limits Programmatically

Applications can programmatically determine current application limits - this needs to be further defined

8. Faults

API calls that return an error return one of the following fault objects. All fault objects extend from the base fault, `serviceFault`, for easier exception handling for languages that support it. Again these are probably common to all OpenStack APIs

Chapter 4. API Operations

This chapter explains specific API operations. For ideas relevant to all API operations, see the "General API Information" chapter.

1. Load Balancers

1.1. List Load Balancers

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

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.

To view deleted load balancers, add "?status=DELETED" to the end of the get url. A deleted load balancer is immutable and irrecoverable. Only a limited set of attributes will be returned in the response object:

- `id`
- `name`
- `created`
- `updated`

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":"2010-11-30T03:23:42Z"
            },
            "updated":{
                "time":"2010-11-30T03:23:44Z"
            }
        },
        {
            "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":"2010-11-30T03:23:42Z"
            },
            "updated":{
                "time":"2010-11-30T03:23:44Z"
            }
        }
    ]
}
```

1.2. List Load Balancer Details

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

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

```
    "loadBalancer":{
        "id": 2000,
        "name":"sample-loadbalancer",
        "protocol":"HTTP",
        "port": 80,
        "algorithm":"RANDOM",
        "status":"ACTIVE",
        "connectionLogging":{
            "enabled":"true"
        },
        "virtualIps":[
            {
                "id": 1000,
                "address":"206.10.10.210",
                "type":"PUBLIC",
                "ipVersion":"IPV4"
            }
        ],
        "nodes":[
            {
                "id": 1041,
                "address":"10.1.1.1",
                "port": 80,
                "condition":"ENABLED",
                "status":"ONLINE"
            },
            {
                "id": 1411,
                "address":"10.1.1.2",
                "port": 80,
                "condition":"ENABLED",
                "status":"ONLINE"
            }
        ],
        "sessionPersistence":{
            "persistenceType":"HTTP_COOKIE"
        },
        "connectionThrottle":{
            "maxRequestRate": 50,
            "rateInterval": 60
        },
        "cluster":{
            "name":"c1.dfw1"
        },
        "created":{
            "time":"2010-11-30T03:23:42Z"
        },
        "updated":{
            "time":"2010-11-30T03:23:44Z"
        }
    }
}
```

1.3. 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 has a length that is configurable and can be queried when querying limits.

Note

Users may configure all documented features of the load balancer at creation time by simply providing the additional elements or attributes in the request. This document provides an overview of all the 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"
            }
        ]
    }
}
```

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 or DNS name. For example, this method makes it possible to use the same load balancing configuration to support 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.demo.net" />
    <created time="2011-02-08T21:19:55Z" />
    <updated time="2011-02-08T21:19:55Z" />
    <connectionLogging enabled="false" />
</loadBalancer>
```

Example 4.10. Create Load Balancer (Required Attributes with Shared IP) Response: JSON

```
{
    "loadBalancer": {
        "name": "a-new-loadbalancer",
        "id": 144,
        "protocol": "HTTP",
        "port": 83,
        "algorithm": "RANDOM",
        "status": "BUILD",
        "cluster": {
            "name": "ztm-n01.staging1.lbaas.demo.net"
        },
        "nodes": [
            {
                "address": "10.1.1.1",
                "id": 653,
                "port": 80,
                "status": "ONLINE",
                "condition": "ENABLED",
                "weight": 1
            }
        ],
        "virtualIps": [
            {
                "address": "206.10.10.210",
                "id": 39,
                "type": "PUBLIC",
                "ipVersion": "IPV4"
            }
        ],
        "created": {
            "time": "2011-04-13T14:18:07Z"
        },
        "updated": {
            "time": "2011-04-13T14:18:07Z"
        },
        "connectionLogging": {
            "enabled": false
        }
    }
}
```

1.4. 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 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`

This operation does not return a response body.

Note

The load balancer's ID and status are immutable attributes (do we need to add port and and protocol here as well?) Leaving them in for now and cannot be modified by the caller. Supplying an unsupported attribute will result in a 400 (badRequest) fault.

Example 4.11. 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 4.12. Update Load Balancer Attributes Request: JSON

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

All load balancers also 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 while 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 4.1. 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 load balancer; contact Support.
DELETED	Load balancers in DELETED status can be displayed for a certain number of days after deletion.The number of days is configuraable

1.5. Remove Load Balancer

Verb	URI	Description
DELETE	/loadbalancers/*loadBalancerId*	Remove a load balancer from the 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.

2. Nodes

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. Only nodes that are in 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. The "draining node" can be added as an extension. If the traffic manager receives a requestand 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.

The caller can assign weights to the nodes as part of the weight attribute of the node element. When 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 condition and removing it.

2.1. List Nodes

Verb	URI	Description
GET	/loadbalancers/*loadBalancerId*/nodes	List node(s) configured for the load balancer.
GET	/loadbalancers/*loadBalancerId*/nodes/*nodeId*	List details of a specific node.

Normal Response Code(s): 200, 202

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

This operation does not require a request body.

Example 4.13. List Node Response: XML

```
<node
    id="410"
    address="10.1.1.1"
    port="80"
    condition="ENABLED"
    status="ONLINE"
    weight="12" />
```

Example 4.14. List Node Response: JSON

```
{"node": {
        "id":"410",
        "address":"10.1.1.1",
        "port":80,
        "condition":"ENABLED",
        "status":"ONLINE",
        "weight":12
    }
}
```

Example 4.15. List Nodes Response: XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
    <node
            id="650"
            address="10.1.1.1"
            port="80"
            condition="ENABLED"
            status="ONLINE"/>
    <node
            id="183"
            address="10.2.2.1"
            port="80"
            condition="ENABLED"
            status="ONLINE"/>
    <node
            id="184"
            address="10.2.2.2"
            port="88"
            condition="ENABLED"
            status="ONLINE"/>
</nodes>
```

Example 4.16. List Nodes Response: JSON

```
{
    "nodes": [
        {
            "address": "10.1.1.1",
            "id": 650,
            "port": 80,
            "status": "ONLINE",
            "condition": "ENABLED"
        },
        {
            "address": "10.2.2.1",
            "id": 183,
            "port": 80,
            "status": "ONLINE",
            "condition": "ENABLED"
        },
        {
            "address": "10.2.2.2",
            "id": 184,
            "port": 88,
            "status": "ONLINE",
            "condition": "ENABLED"
        }
    ]
}
```

2.2. Add Nodes

Verb	URI	Description
POST	/loadbalancers/*loadBalancerId*/nodes	Add a new node to the load balancer.

Normal Response Code(s): 200, 202

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

This operation does not require a request body.

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

Example 4.17. Add Nodes Request: XML

```
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
    <node address="10.1.1.1" port="80" condition="ENABLED" />
    <node address="10.2.2.1" port="80" condition="ENABLED" />
    <node address="10.2.2.2" port="88" condition="ENABLED" weight="10"/>
</nodes>
```

Example 4.18. Add Nodes Request: JSON

```
{"nodes": [
        {
            "address": "10.1.1.1",
            "port": 80,
            "condition": "ENABLED"
        },
        {
            "address": "10.2.2.1",
            "port": 80,
            "condition": "ENABLED"
        },
        {
            "address": "10.2.2.2",
            "port": 88,
            "condition": "ENABLED",
            "weight": 10
        }
    ]
}
```

Example 4.19. Add Nodes Response: XML

```
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
    <node
        address="10.2.2.3"
        id="185"
        port="80"
        condition="ENABLED"
        status="ONLINE"
        weight="1" />
    <node
        address="10.2.2.4"
        id="186"
        port="80"
        condition="ENABLED"
        status="ONLINE"
        weight="1" />
</nodes>
```

Example 4.20. Add Nodes Response: JSON

```
{"nodes": [
        {
            "address": "10.2.2.3",
            "id": 185,
            "port": 80,
            "status": "ONLINE",
            "condition": "ENABLED",
            "weight": 1
        },
        {
            "address": "10.2.2.4",
            "id": 186,
            "port": 80,
            "status": "ONLINE",
            "condition": "ENABLED",
            "weight": 1
        }
    ]
}
```

2.3. Modify Nodes

Verb	URI	Description
PUT	/loadbalancers/*loadBalancerId*/nodes/*nodeId*	Modify the configuration of a node on the load balancer.

Normal Response Code(s): 200, 202

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

This operation does not require a request body.

Note

The node's IP and port are immutable attributes and cannot be modified with a **PUT** request. Supplying an unsupported attribute will result in a 400 (badRequest) fault. A load balancer supports a maximum number of nodes. The number of nodes is a configurable property; the maximum weight of a node is 100.

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

Table 4.2. 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.

Example 4.21. Update Node Condition Request: XML

```
<node condition="ENABLED" weight="12" />
```

Example 4.22. Update Node Condition Request: JSON

```
{"node":{
        "condition": "ENABLED",
        "weight": 59
    }
}
```

2.4. Remove Nodes

Verb	URI	Description
DELETE	/loadbalancers/*loadBalancerId*/nodes/*nodeId*	Remove a node from the load balancer.

Normal Response Code(s): 200, 202

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

This operation does not require a request body.

3. Virtual IPs

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 private address, routable only within the region in which the load balancer resides.

Table 4.3. Virtual IP Types

Name	Description
PUBLIC	An address that is routable on the public Internet.
INTERNAL	An address that is routable only on internal network.

3.1. List Virtual IPs

Verb	URI	Description
GET	/loadbalancers/*loadBalancerId*/virtualips	List all virtual IPs associated with a load balancer.

Normal Response Code(s): 200, 202

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

This request does not require a request body.

Example 4.23. 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 4.24. List Virtual IPs Response: JSON

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

3.2. Remove Virtual IP

Verb	URI	Description
DELETE	/ loadbalancers/*loadBalancerId*/ *virtualIpId*	Remove a virtual IP.

Normal Response Code(s): 200, 202

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

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.

4. Usage Reports

4.1. List Usage

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

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.

The load balancer usage reports provide a minimumum set of usage counters. This list will definded such that it is supported by differen load balancing technologies. Values to be considered are incoming/outgoing transfer of bytes.

5. Monitors

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. A loadbalancer will have a configurable amount of health monitors at a time.

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

Table 4.4. Health Monitor Types

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

5.1. Monitor Health

Verb	URI	Description
GET	/loadbalancers/*`loadBalancerId`*/healthmonitor	Retrieve the health monitor configuration, if one exists.
PUT	/loadbalancers/*`loadBalancerId`*/healthmonitor	Update the settings for a health monitor.
DELETE	/loadbalancers/*`loadBalancerId`*/healthmonitor	Remove the health monitor.

Normal Response Code(s): 200, 202

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

5.1.1. Monitor Connections

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 checking the health of a node after it is put into OFFLINE status.

- `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. (Must be a number between 1 and 10)

Example 4.25. Monitor Connections Request: XML

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

Example 4.26. Monitor Connections Request: JSON

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

Example 4.27. Monitor Connections Response: XML

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

Example 4.28. Monitor Connections Response: JSON

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

5.1.2. Monitor HTTP and HTTPS

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

- `delay`: The minimum number of seconds to wait before executing the health monitor. (Must be a number between 1 and 3600)
- `timeout`: The maximum number of seconds to wait for a connection to be established before timing out. (Must be a number between 1 and 300)
- `attemptsBeforeDeactivation`: The number of permissible monitor failures before removing a node from rotation. (Must be a number between 1 and 10)
- `path`: The HTTP path that will be used in the sample request. The most basic proposal could be to support "path" only and assume that method is always "GET" and that status code is expected to be always 200 to declare probe success. This will be widely supported but is not very powerful. Needs researching

Example 4.29. Monitor HTTP Response: XML

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

Example 4.30. Monitor HTTPS Response: XML

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

6. Sessions

6.1. Manage Session Persistence

Session persistence is a feature of the load balancing service which forces multiple 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 4.5. 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.

Verb	URI	Description
GET	/loadbalancers/loadBalancerId/sessionpersistence	List session persistence configuration.

Verb	URI	Description
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)

Example 4.31. List Session Persistence Configuration Response: XML

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

Example 4.32. List Session Persistence Configuration Response: JSON

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

Example 4.33. Set Session Persistence Type Request: XML

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

Example 4.34. Set Session Persistence Type Request: JSON

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

7. Connections

7.1. Log Connections

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 view the current connection logging configuration, enable connection logging, or disable connection logging.

This operation does not require a request body.

Example 4.35. List Connection Logging Configuration Response: XML

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

Example 4.36. List Connection Logging Configuration Response: JSON

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

Example 4.37. Enable Connection Logging Request: XML

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

Example 4.38. Enable Connection Logging Request: JSON

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

7.2. Throttle Connections

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)

This operation does not require a request body.

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.

- `maxRequestRate`: Maximum number of requests allowed from a single IP address in the defined `rateInterval`. Setting a value of 0 allows an unlimited connection rate; otherwise, set a value between 1 and 100000.
- `rateInterval`: Frequency (in seconds) at which the `maxReqestRate` is assessed. For example, a `maxRequestRate` of 30 with a `rateInterval` of 60 would allow a maximum of 30 connections per minute for a single IP address. This value must be specified between 1 and 3600.

Example 4.39. List Connection Throttling Configuration Response: XML

```
<connectionThrottle xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
    maxRequestRate="50"
    rateInterval="60" />
```

Example 4.40. List Connection Throttling Configuration Response: JSON

```
{"connectionThrottle":{
    "maxRequestRate": 50,
    "rateInterval": 60
  }
}
```

Example 4.41. Update Connection Throttling Configuration Request: XML

```
<connectionThrottle xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
    maxRequestRate="50"
    rateInterval="60" />
```

Example 4.42. Update Connection Throttling Configuration Request: JSON

```
{
    "maxRequestRate": 50,
    "rateInterval": 60
}
```

8. Protocols

8.1. List 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)

This operation does not require a request body.

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 default port for the given protocol will be selected unless otherwise specified.

Example 4.43. 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 4.44. 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"
        }
    ]
}
```

9. Algorithms

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. It was decided that ROUND-ROBING would be the default protocol. If an implementation doesn't support a protocol, it should return "Invalid value"

Table 4.6. 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. This should be the default proctocol
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.

9.1. List 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)

This operation does not require a request body.

Example 4.45. 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 4.46. List Load Balancing Algorithms Response: JSON

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

Chapter 5. API Extensions

Implementations of the API specifications are free to augment with extensions as they see appropriate to extend Load Balancing features (e.g. support for new LB algorithms) in this API or offer new ones.All client applications written to this core specification (using the base API) should work against extended implementation as specified in this document. Therefore, applications should not receive values not specified in this specification or obtain a different behavior than expected when they are not using (or aware of the availability of) extended APIs.

1. Discovering extentions

Implementations should allow users to discover if the LB service supports extensions. together with details about the extensions(document pointers, etc.).

2. Using extended APIs

Clients should operate on resources using extension URIs in order to make use of the extension URI.

POST `/loadbalancers/ext/<ext-name>`

The sames load-balancer cand be retrieved using either: **GET** `/loadbalancers/loadBalancerId` in which case, only the deatures and attributes in this document are returned OR GET/loadbalancer/ex/ext-name/loadBalancerId which returns the extended version of the loadbalancer (with extra attributes).

Clients can always use the core APIs for resources created or updated with extension APIs. Extension resources are always defined as pure supersets of core resources. New types of resources can be added in the extension API that have no counterpart in the core API. These resouces can be used only from the extension API.