Load Balancing Service API Specification

11/11/10

Table of Contents

Intended Audience

This guide is intended for software developers who want to develop applications using the Load balancing Service API. It assumes the reader has a general understanding of the Load Balancing service and is familiar with:

- RESTful Web Services
- HTTP/1.1 Conventions
- JSON and/or Serialization Formats
- Atom Syndication Format

Concepts and Definitions

The load balancing API specification has several key concepts that are important to understand.

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 the configuration.

Virtual IP

A virtual IP is an 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.

Node

A nodes is a back-end device providing a service on a given IP and port.

Health Monitor

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

Passive Health Monitor

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

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

If the tests above 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 for HTTP traffic or the connection will be dropped.

Note: If a given node fails a passive health monitor check for three consecutive times, then the service assumes the node has failed and will place it in the 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.

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, and https. These probes are run at configured intervals and in the event of a failure, the node status changes to OFFLINE and the node will not receive traffic. Once the probe has detected that the node has recovered (by running a subsequent test), then the node is changed to the ONLINE status and will be capable of servicing requests.

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 its online.

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 your Cloud Files account. Logs are delivered to your account every hour.

General API Information

The Load Balancing Service 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 SSL over HTTP (HTTPS) on TCP port 443.

Authentication

Every REST request against the load balancing service requires the inclusion of a specific authorization token, which is 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. In order to authenticate, you must supply your username and API access key in the X-Auth-Key HTTP headers. Username is your common Rackspace Cloud username and your API access key can be obtained from the Rackspace Cloud Control Panel in the Your Account | API Access section.

Sample 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 as noted above.

Sample 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/
v1.0/35428
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 401 Unauthorized response from a service endpoint.

Note

LBaaS will depend on Cloud Auth 1.1, which will include the endpoints in the response payload - To be updated once contract is solidified with auth developers.

Service Access

The load balancing service is a regionalized service, which allows for the caller to select a region to provision a load balancer into. In the response payload provided by the authentication service, the caller can choose the appropriate region-based endpoint.

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 in. 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.

Request/Response Types

The load balancing 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 (see example below). 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.

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.

Persistence 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.

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 parameters limit and offset can be set in the URI (e.g.? limit=0&offset=0). If an offset is given beyond the end of a list an empty list will be returned. Note that list operations never return itemNotFound (404) faults.

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 will be returned. If data has changed, only the items changed since the specified time will be returned in the response. 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.

API Operations

List Load Balancers

Normal Response Code(s): 200

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

Description

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

Deleted Load Balancers

Load balancer which have been deleted will be shown in this list until all usage has been processed and invoiced. A deleted load balancer is immutable and irrecoverable. Only a limited set of attributes will be returned in the response object (id, name, status, created, and updated).

Request

This operation does not require a request body.

Sample XML Response (/loadbalancers)

```
<?xml version="1.0" ?>
<loadBalancers xmlns="http://docs.rackspacecloud.com/
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>
```


Sample Json Response (/loadbalancers)

```
{"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"
         }
      }
   ]
```



```
}
```

List Load Balancer Details

Normal Response Code(s): 200

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

Description

This operation provides the 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.

Request

This operation does not require a request body.

Sample XML Response

```
<loadBalancer xmlns="http://docs.rackspacecloud.com/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>
```

Sample JSON Response

```
{
  "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":{
      "minConnections":"10",
      "maxConnections":"100",
      "maxConnectionRate":"50",
      "rateInterval":"60"
    },
    "cluster":{
      "name":"c1.dfw1"
    },
    "created":{
      "time":"2010-11-30T03:23:42.000+0000"
    },
    "updated":{
      "time":"2010-11-30T03:23:44.000+0000"
    }
  }
}
```

Create Load Balancer

Normal Reponse Code(s): 202

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

Description

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 (for more details on this operation see the "List Load Balancer Details" section of this document). 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.

Sharing Virutal IPs Between Load Balancers

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 virtualIp ID instead of a type. (see below for a sample request). 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.

Sample XML Request (Required Attributes Only)

```
<loadBalancer xmlns="http://docs.rackspacecloud.com/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>
```

Sample JSON Request (Required Attributes Only)

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


Sample XML Request (Required Attributes Only w/ Shared IP)

```
<loadBalancer xmlns="http://docs.rackspacecloud.com/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>
```

Sample JSON Request (Required Attributes Only w/ Shared IP)

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

Sample XML Response

```
<loadBalancer xmlns="http://docs.rackspacecloud.com/loadbalancers/
api/v1.0"
   id="1041"
   name="a-new-loadbalancer"
   status="BUILD"
```



```
   port="80"
   protocol="HTTP">
   <virtualIps>
      <virtualIp
         id="1000"
         address="206.10.10.210"
         type="PUBLIC" />
   </virtualIps>
   <nodes>
      <node address="10.1.1.1" port="80" condition="ENABLED" />
   </nodes>
</loadBalancer>
```

Remove Load Balancer

Normal Response Code(s): 202

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

Description

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.

Request

This operation does not require a request body.

Response

This operation does not return a response body.

List, Add, Modify, and Remove Nodes

Normal Response Code(s): 20, 202

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

Description

The nodes defined by the load balancer are responsible for servicing the requests received through the load balancers 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 (for more information on health check settings, see the "Health Check" section of this document) If a back-end node is not listening on its port or does not meet the conditions

of the defined 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"* stats will receive and be able to service traffic from the load balancer.

All nodes have an associated status that indicates if 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 more 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 either by the passive or active health monitors. For more information on health monitoring, see the concepts and definitions section (above).

If the weighted round robin load balancer algorithm mode is selected (WEIGHTED_ROUND_ROBIN), 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 and as a result, nodes can either be internal ServiceNet addresses or addresses on the public Internet.

Request (GET)

This operation does not require a request body.

Sample XML Response (Get)

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

Sample XML Request (POST)(Required attributes only)

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


Sample XML Response (POST)

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

Sample JSON Request (POST)(Required attributes) (/loadbalancers/loadBalancerId/nodes)

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

Sample JSON Response (POST)

```
{"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 badRequest (400) fault.

Sample XML Request (PUT) (/loadbalancers/loadBalancerId/nodes/nodeId) Mutable Attributes Only


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

Sample JSON Request (PUT) (/loadbalancers/loadBalancerId/nodes/nodeId)

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

Response (PUT)

This operation does not return a response body.

Request (DELETE)

This operation does not require a method body.

Response (DELETE)

This operation does not return a response body.

Update Name, Algorithm, Protocol, Port, and/or Maximum Concurrent Connection Attributes of a Load Balancer

Normal Response Code(s): 202

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

Description

This operation asynchronously updates the attributes of a given 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

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 badRequest (400) fault.

Sample XML (PUT)(Mutable Attributes Only)

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

Sample JSON (PUT)(Mutable Attributes Only)

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

Response

This operation does not return a response body.

Virtual IP Management

Normal Response Code(s): 200, 202

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

Description

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

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

Virtual IP Types

Request (GET)

This request does not require a request body.

Sample XML GET Response (/loadbalancers/loadBalancerId/virtualips)

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

Sample JSON XML Response (/loadbalancers/loadBalancerId/virtualips)

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

Request (DELETE)

This operation does not require a request body.

Response (DELETE)

This operation does not return a response body.

Usage Reporting

Normal Response Code(s): 200

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

Description

The service usage provides current and a historical view of tier profiles and transfer associated with the load balancing service. Current usage represents any usage that been

incurred since the endDate of the most recent historical usage object. Current usage is processed into historical usage objects every 24 hours or whenever an event occurs that changes a billing-related attribute (for example, changing the tier profiles of a load balancer). Values for both incomingTransfer and outgoingTransfer are expressed in bytes transferred.

Note
Historical usage is available for up to 90 days of service activity.

Request

This operation does not require a request body.

Sample XML Response (GET - for all usage)

```
<loadBalancerUsage xmlns="http://docs.rackspacecloud.com/
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"/>
  <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"/>
  <loadBalancerUsageRecord id="476" 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>
```

Sample JSON Response (GET)

```
{
    "loadBalancerUsage": [
        {
            "id": "1",
            "averageNumConnections": "12.45",
            "incomingTransfer": "323425",
            "outgoingTransfer": "6934400",
            "numVips": "1",
```



```
            "numPolls" "32",
            "startTime": "2010-02-01T05:00:00Z",
            "endTime": "2010-02-01T06:00:00Z"
        },
        {
            "id": "2",
            "averageNumConnections": "16.78",
            "incomingTransfer": "95109",
            "outgoingTransfer": "81401088",
            "numVips": "2",
            "numPolls": "5",
            "startTime": "2010-01-01T07:57:00Z",
            "endTime": "2010-01-01T08:57:00Z"
        }
    ]
}
```

Sample XML Response (/loadbalancers/usage?startDate=2010-11-30&endDate=2010-12-01)

```
<accountBilling xmlns="http://docs.rackspacecloud.com/loadbalancers/api/v1.0" accountId="1106">
 <accountUsage>
  <accountUsageRecord numLoadBalancers="2" numPublicVips="1" numServicenetVips="0" startTime="
 </accountUsage>
 <loadBalancerUsage loadBalancerId="66" loadBalancerName="My first loadbalancer">
  <loadBalancerUsageRecord id="394" averageNumConnections="0.0" incomingTransfer="0" outgoingTr
     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" outgoingTr
     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" outgoingTr
     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" outgoingTr
     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" outgoingTr
    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" outgoingTra
     startTime="2010-12-21T12:32:07-06:00" endTime="2010-12-21T12:36:30-06:00"/>
 <loadBalancerUsage/>
</accountBilling>
```

Access List Management

Normal Response Code(s): 200, 202

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

Description

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 *DENY* type will always take precedence over items with the *ALLOW* 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 replaced with subsequent POST requests. To append to the access list, a PUT request should be issued. Like the POST operation the PUT operation requires one or more network items. 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.

Request (GET)

This operation does not require a request body.

Sample XML Response (GET)

```
<accessList xmlns="http://docs.rackspacecloud.com/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>
```

Sample JSON Response (GET)

```
{
    "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"
        }
    ]
}
```

Sample XML Request (POST)(Required attributes)

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

Sample JSON Request (POST)(Required attributes)


```
{
    "networkItems": [
        {
            "address": "206.160.163.21",
            "id": 23,
            "type": "DENY"
        },
        {
            "address": "206.160.165.11",
            "id": 24,
            "type": "DENY"
        }
    ]
}
```

Request (DELETE)

This operation does not require a request body.

Response (DELETE)

This operation does not return a response body.

Response (POST)

This operation does not return a response body.

Active Health Monitoring

Normal Response Code(s): 200, 202

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

Description

The load balancing service includes a health monitoring operation, which periodically checks your backend nodes to ensure they are responding correctly. In the event that the node is not responding, it will be removed from rotation until the health check determines the node is functional. In addition to being run at periodic intervals, the health check also is run against every node that is added to ensure it 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. An overview of the supported health monitors are detailed below:

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, which are detailed below.

- *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.

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.

Health Monitor Types

All health monitors have a type attribute to signify what kind of monitor it is.

Sample XML Response (GET) [Connect Monitor]

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

Sample JSON Response (GET) [Connect Monitor]

```
{
    "type": "CONNECT",
    "delay": "10",
    "timeout": "10",
```



```
    "attemptsBeforeDeactivation": "3"
}
```

Sample XML Response (GET) [HTTP Monitor]

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

Sample XML Response (GET) [HTTPS Monitor]

```
<healthMonitor xmlns="http://docs.rackspacecloud.com/
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.

Sample XML Request (PUT) [Connect Monitor]

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

Sample JSON (PUT) [Connect Monitor]


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

Response (POST)

This operation does not return a response body.

Response (PUT)

This operation does not return a response body.

Response (PUT)

This operation does not require a request body.

Response (DELETE)

This operation does not return a response body.

Session Persistence

Normal Response Code(s): 200, 202

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

Description

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 inheritly share application state between backend servers.

Session Persistence Modes

Sample XML Response (GET)

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

Sample JSON Response

```
{
   "sessionPersistence":{
```



```
      "persistenceType":"HTTP_COOKIE"
   }
}
```

Sample XML Request (POST)

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

Sample JSON Request (POST)

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

Response (POST)

This operation does not return a response body.

Request (DELETE)

This operation does not require a request body.

Response (DELETE)

This operation does not return a response body.

Connection Logging

Normal Response Code(s): 200, 202

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

Description

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

Request (GET)

This operation does not require a request body.

Sample XML Response (GET)

```
```



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

Sample JSON Response (GET)

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

Sample XML Request (PUT)

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

Sample JSON Request (PUT)

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

Connection Throttling

Normal Response Code(s): 200, 202

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

Description

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 the maxConnectionRate is assessed (in seconds). 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.

Request (GET)

This operation does not require a request body.

Sample XML Response (GET)

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

Sample JSON Response (GET)

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

Note
Attributes that are required are displayed in the samples unless otherwise stated.

Sample XML Request (PUT)

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

Sample JSON (PUT)

```
{
    "maxConnections": 10,
    "minConnections": 100,
```



```
    "maxConnectionRate": 50,
    "rateInterval": 60
}
```

Response (PUT)

This operation does not return a response body.

Request (DELETE)

This operation does not require a request body.

Response (DELETE)

This operation does not return a response body.

Load Balancing Protocols

Normal Response Code(s): 200

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

Description

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 backend nodes. When configuring a load balancer, the port selected will be the default port for the given protocol unless otherwise specified.

Request

This operation does not require a request body.

Sample XML Response

```
<protocols xmlns="http://docs.rackspacecloud.com/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>
```


Sample JSON Response


```
{"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"
      }
   ]
}
```


Load Balancing Algorithms

Normal Response Code(s): 200

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

Description

All load balancers utilize a "load balancer algorithm" that defines how traffic should be directed between backend nodes. The default algorithm for newly created load balancers is Random, which can be overrided 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.

Request

This operation does not require a request body.

Sample XML Response

```
<algorithms xmlns="http://docs.rackspacecloud.com/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>
```

Sample JSON Response

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


Load Balancing Algorithms Overview

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 being made, but have not yet been finalized. Load balancers in a suspended status are configured to reject traffic and will not forward requests to backend nodes.

Node Condition

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

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.

serviceFault

The serviceFault and by extension all other faults shall contain a message and detail element containing 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.

loadBalancerFault

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

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

badRequest

The operation was not a valid request. This fault indicates that the data in the request object is invalid, for example a string was used on a parameter that was expecting an integer. The fault will wrap validation errors.

```
<badRequest xmlns="http://docs.rackspacecloud.com/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>
```

itemNotFound


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

overLimit

This fault is returned when the user had exceeded their currently allocated limit.

```
<overLimit code="500" xmlns="http://docs.rackspacecloud.com/
loadbalancers/api/v1.0">
    <message>Your account is currently over the limit so your
 request could not be processed.</message>
</overLimit>
```

unauthorized

This fault is returned when you attempt to perform an operation you are not authorized to perform.

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

outOfVirtualIps

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

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

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.rackspacecloud.com/
loadbalancers/api/v1.0">
    <message>The object at the specified URI is immutable and can
 not be overwritten.</message>
</immutableEntity>
```

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.rackspacecloud.com/loadbalancers/api/v1.0">
    <message>The Object at the specified URI is unprocessable.</
message>
</unprocessableEntity>
```

serviceUnavailable

This fault is returned when the service is unavailable. For example, if the service is undergoing maintenance or otherwise unavailable. Note that this does not necessarily mean the currently configured loadbalancers are unable to process traffic, It simply means that the API is currently unable service requests.

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

