Mapping of Atlas 1.1 API to some loadbalancing products

1. Contents

2. Overview

This document demonstrates the mapping of the OpenStack Load Balancing core API proposal to the equivalent configurations on 3 widely used load balancers:

- HA Proxy
- Citrix NetScaler
- Nginx

2.1. *HA Proxy*

HAProxy is an open source load balancer that can run as a daemon on Linux.

Because HAProxy has a limited CLI mode (using *Unix stats sockets*) and doesn't have an API, all configuration operations for mapping the OpenStack LB API would be performed through editing the HAProxy configuration file and restarting the HAProxy daemon.

2.2. *NetScaler*

NetScaler is a Citrix product that comes in the form of hardware appliances (NetScaler MPX range) and soft appliances (NetScaler VPX).

The NetScaler product has a long list of L2 to L7 features beyond load balancing, such as SSL Offload, AppFirewall, Caching, Compression, Rewriting engine, VPNs, and many others.

All NetScaler features can be configured either through the CLI, a Web GUI interface or using a SOAP (XMLAPI) or REST (Nitro) API.

The Load Balancing feature is configured by creating LB vservers and binding service groups to them (representing backend nodes).

In this document, we will use CLI commands for brevity to show the mapping from the OpenStack API, even though NetScaler's REST API (Nitro) will be more appropriate to use in an implementation of the LB service.

2.3. *Nginx*

Nginx is a high-performance and lightweight open source web server that can be augmented with modules to perform load-balancing, SSL offloading, content redirection, etc. The main modules required for mapping the LB API are *Proxy*, *Upstream, Log, StubStatus, HttpLimitReqModule*, *ngx_http_healthcheck_module, nginx_tcp_proxy_module, HttpSslModule, MailSslModule* modules. Nginx needs to be compiled with these modules and started with the command-line options to load them.

Nginx doesn't have a CLI mode or an API. All configuration operations are done by editing the configuration file and restarting the Nginx master process.

3. API Operation Mappings

3.1. *List LoadBalancers*

Verb	URI	Description	Representation
GET	/loadbalancers	List all loadbalancers configured for an account	XML, JSON

Example: List Load Balancers Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<loadBalancers xmlns="http://docs.openstack.org/loadbalancers/api/v1.0">
  <loadBalancer id="71" name="lb-site1" status="ACTIVE"
    protocol="HTTP"  port="80" algorithm="ROUND_ROBIN">
    <virtualIp address="206.55.130.2" ipVersion="IPV4" type="PUBLIC" />
    <cluster name="dc-eastcoast-4" />
    <created time="2010-11-29T17:31:41Z" />
    <updated time="2010-11-30T08:35:15Z" />
  </loadBalancer>
  <loadBalancer id="166"  name="lb-site2" status="ACTIVE"
    protocol="HTTP" port="80" algorithm="LEAST_CONNECTIONS">
    <virtualIp address="206.55.130.15" ipVersion="IPV4" type="PUBLIC" />
    <cluster name="dc-eastcoast-4" />
    <created time="2010-11-30T03:23:42Z" />
    <updated time="2010-11-30T03:23:44Z" />
  </loadBalancer>
</loadBalancers>
```

HA Proxy

The LB service will need to find all the "listen" or "frontend" sections in the HAProxy configuration file and parse their content, like the following section.

```
listen lb-3271 206.55.130.2:80
       mode http
       …

       …
```

NetScaler (CLI)

The LB service uses the "show lb vserver" CLI command to returns details all lbvservers on NetScaler, which can be filtered for those belonging to the user account.

```
show lb vserver
```

Nginx

The LB service will need to find all the "server" sections in the Nginx configuration file that have a proxy_pass directive, and parse their content, like the following section.

```
http {
      …
      #lb-3271
      server {
         listen 206.55.130.2:80
         server_name .*;
         location / {
            proxy_pass …;
         }
      }
      …
}
```

3.2. *List LoadBalancer Details*

Verb	URI	Description	Representation

GET	/loadbalancers/*loadBalancerId*	List details of the specified load balancer	XML, JSON

Example: List Load Balancer Details Response: XML

HTTP Request: `GET /loadbalancers/71`

HTTP Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/v1.0" id="71" name="lb-site1"
status="ACTIVE" protocol="HTTP"  port="80" algorithm="ROUND_ROBIN">
   <virtualIp address="206.55.130.2" ipVersion="IPV4" type="PUBLIC" />
   <nodes>
      <node  id="1041" address="10.1.1.1" port="80" condition="ENABLED"  status="ONLINE" />
      <node  id="1411" address="10.1.1.2" port="80" condition="ENABLED"  status="ONLINE" />
   </nodes>
   <sessionPersistence    persistenceType="HTTP_COOKIE"  />
   <connectionLogging enabled="true" />
   <cluster name="c1.dfw1" />
   <created time="2010-11-30T03:23:42Z" />
   <updated time="2010-11-30T03:23:44Z" />
</loadBalancer>
```

HA Proxy

The LB service will need to find the "listen" section that corresponds to the requested load balancer.

```
listen lb-4532 206.55.130.2:80
       mode http
```

The listen section should include all the settings, including those for the nodes of the load balancer which would be represented by the "server" directives in these sections.

NetScaler (CLI)

On NetScaler, the LB service uses the "show lb vserver <name>" CLI command to return the lb vserver that corresponds to the load balancer. It is the responsibility of the LB service to decide how to maintain a mapping between the load balancer id and the resources on NetScaler such as the corresponding lb vserver.

```
show lb vserver lb-4532
```

From the previous command, one can also find out the services bound to the lbvserver. For each service, one can get the details of the service (node) using the "show service" command:

```
show service node-1421

show service node-627
```

Nginx

The LB service will need to find the "server" section that corresponds to the requested load balancer.

```
http {
      #lb-4532
      server {
         listen 206.55.130.2:80
         …
      }
}
```

The listen section should include all the settings, including those for the nodes of the load balancer which would be represented by the "server" directives in these sections.

3.3. *Create LoadBalancer*

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

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

HTTP Request: `POST /loadbalancers`

HTTP Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<loadBalancer xmlns="http://docs.openstack.org/loadbalancers/api/v1.0" id="71" name="a-new-loadbalancer" protocol="HTTP"  port="80" algorithm="ROUND_ROBIN">
   <virtualIp  type="PUBLIC" />
   <nodes>
      <node   address="10.1.1.1" port="80"  />
      <node   address="10.1.1.2" port="8080"   />
   </nodes>
</loadBalancer>
```

HA Proxy

The LB service needs to allocate a Virtual IP from its "PUBLIC" pool of IPs, and configure a "listen" section in the HAProxy configuration file that corresponds to the created load balancer.

Example

```
listen a-new-loadbalancer 206.55.130.2:80
       mode http
       balance roundrobin
       server 71-node1 10.1.1.1:80
       server 71-node2 10.1.1.2:8080
```

The LB service will also need to allocate an ID to the newly created load balancer and associate the resources created on HAProxy with this ID.

NetScaler (CLI)

The LB service needs to allocate a Virtual IP from its "PUBLIC" pool of IPs and then use the "add lb vserver" and "add service" CLI commands to create an lbvserver and services (nodes), and then bind the services to the lbvserver.

```
add lb vserver a-new-loadbalancer http 206.55.130.2 80 -lbmethod
roundrobin

add service 71-node1  10.1.1.1 80

add service 71-node2 10.1.1.2 8080

bind lb vserver  a-new-loadbalancer  71-node1

bind lb vserver  a-new-loadbalancer  71-node2
```

The LB service will also need to allocate an ID to the newly created load balancer and associate the resources created on NetScaler with this ID.

Nginx

The LB service needs to allocate a Virtual IP from its "PUBLIC" pool of IPs, and configure a "server" section in the Nginx configuration file that corresponds to the created load balancer.

Example of creating an HTTP load balancer

```
http {
      …
      #a-new-loadbalancer
      server {
         listen 206.55.130.2:80
         location / {
            proxy_pass http://a-new-loadbalancer-nodepool;
         }
       }

      upstream  a-new-loadbalancer-nodepool {
         server   10.1.1.1:80;   #71-node1
         server   10.1.1.2:8080; #71-node2
      }
      …
}
```

Example of creating a TCP load balancer

```
tcp {
        #a-new-loadbalancer
        server {
            listen 206.55.130.2:3721;
            proxy_pass a-new-loadbalancer-nodepool;
        }


        upstream a-new-loadbalancer-nodepool {
            # simple round-robin
            server   10.1.1.1:3721; #71-node1
            server   10.1.1.2:3722; #71-node2
        }

}
```

The LB service will also need to allocate an ID to the newly created load balancer and associate the resources created on Nginx with this ID.

3.4. *Remove Load Balancer*

Verb	URI	Description
DELETE	/loadbalancers/*loadBalancerId*	Removes a loadbalancer from an account

Example: Delete Load Balancer Request

HTTP Request: `DELETE /loadbalancers/71`

HA Proxy

The LB service will locate and remove the "listen" section in the HAProxy configuration file that corresponds to the load balancer.

```
listen a-new-loadbalancer 206.55.130.2:80
        mode http
        balance roundrobin
        server 71-node1 10.1.1.1:80
        server 71-node2 10.1.1.2:8080
```

NetScaler (CLI)

The LB service locates the name of the lb vserver corresponding to the load balancer, and then uses the "rm lb vserver" and "rm service" CLI commands to remove the lbvserver and its bound services (nodes).

```
rm lb vserver a-new-loadbalancer

rm service 71-node1

rm service 71-node2
```

Nginx

The LB service will locate and remove the "server" section and its corresponding "upstream" section in the Nginx configuration file.

```
#a-new-loadbalancer
server {
    listen 206.55.130.2:80
    location / {
        proxy_pass http://a-new-loadbalancer-nodepool;
    }
}

upstream  a-new-loadbalancer-nodepool {
    server    10.1.1.1:80;    #71-node1
    server    10.1.1.2:8080; #71-node2
}
```

3.5. *List, Add, Modify, and Remove Nodes*

3.5.1. List Nodes

Verb	URI	Description	Representation
GET	/loadbalancers/*loadBalancerId*/nodes	List nodes configured for the load balancer.	XML, JSON

Example: List Node: XML

HTTP Request: `GET /loadbalancers/71/nodes`

HTTP Response Body

```
<?xml version="1.0" encoding="UTF-8"?>
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0" >
   <node  id="1421" address="10.1.1.3" port="80" condition="ENABLED"  status="ONLINE" />
   <node  id="1422" address="10.1.1.4" port="80" condition="ENABLED"  status="ONLINE" />
</nodes>
```

HA Proxy

The LB service will locate the "listen" section corresponding to the load balancer in the HAProxy configuration file and enumerate all the "server" entries in it.

```
listen lb-71 206.55.130.2:80
       mode http
       balance roundrobin
       server node-1421 10.1.1.3:80
       server node-1422 10.1.1.4:80
```

NetScaler (CLI)

The LB service needs to find the name of the lb vserver corresponding to the load balancer, and then uses the "show lb vserver" to retrieve services bound to the lbvsever. It can then fetch the details of each service corresponding to a node of the load balancer using the "show service" command.

```
show lb vserver a-new-loadbalancer

show service node-1421

show service node-1422
```

Nginx

The LB service will locate the "upstream" section corresponding to the load balancer in the HAProxy configuration file and enumerate all the "server" entries in it.

```
http {
      #a-new-loadbalancer
      server {
         listen 206.55.130.2:80
         location / {
            proxy_pass http://a-new-loadbalancer-nodepool;
         }
       }

      upstream  a-new-loadbalancer-nodepool {
         server   10.1.1.1:80;   #71-node1
         server   10.1.1.2:8080; #71-node2
      }
}
```

3.5.2. Add Nodes

Verb	URI	Description	Representation
POST	/loadbalancers/*loadBalancerId*/nodes	Add nodes to the load balancer.	XML, JSON

Example: Add Nodes Request: XML

HTTP Request: `POST /loadbalancers/71/nodes`

HTTP Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<nodes xmlns="http://docs.openstack.org/loadbalancers/api/v1.0" >
  <node  address="10.1.1.5" port="80" />
  <node  address="10.1.1.10" port="80" weight="2" />
</nodes>
```

HA Proxy

The LB service will allocate an ID for each node, and will locate the "listen" section corresponding to the load balancer in the HAProxy configuration file and add "server" entries in it.

```
listen lb-71 206.55.130.2:80
       mode http
       balance roundrobin
       server node-352 10.1.1.5:80
       server node-463 10.1.1.10:80 weight 2
```

NetScaler (CLI)

The LB service will allocate an ID for each node, and uses the "add service" command to create new services. It will then need to locate the lb vserver corresponding to the load balancer and use "bind lb vserver" command to bind the created services to that lbvserver.

```
add service node-352 http 10.1.1.5 80

add service node-463 http 10.1.1.10 80

bind lb vserver lb-71 node-352

bind lb vserver lb-71 node-463 -weight 2
```

Nginx

The LB service will allocate an ID for each node, and will locate the "upstream" section corresponding to the load balancer in the Nginx configuration file and add "server" entries in it.

```
upstream  lb-71-nodepool {
   server   10.1.1.5:80;   #node-352
   server   10.1.1.10:80 weight=2; #node-463
}
```

3.5.3. Modify Node

Verb	URI	Description	Representation
PUT	/loadbalancers/*loadBalancerId*/nodes/*nodeId*	Modifies attributes of a balancer's node.	XML, JSON

Example: Modify Node Request: XML

Request URL: `PUT /loadbalancers/4532/nodes/271`

Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<node xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"   condition=" DISABLED"  />
```

HA Proxy

The LB service will locate the "listen" section corresponding to the load balancer **4532** in the HAProxy configuration file and identify the "server" entry in it that corresponds to the load balancer's node **271**, and modifies it accordingly.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance roundrobin
       server node-271 10.1.1.15:80 weight 4
```

To change the condition of a node in HA Proxy, see section on Node Condition.

NetScaler (CLI)

The LB service uses the "set service" command or "enable/disable service" command to change attributes or state of a service corresponding to a load balancer's node.

```
disable service node-271
```

To change the weight of a node, the LB service needs to locate the service corresponding to the node, change the service binding to the lbvserver corresponding to the load balancer, by removing the existing binding and creating a new one with the desired weight value.

```
unbind lb vserver lb-4532 node-271

bind lb vserver lb-4532 node-271 -weight 4
```

Nginx

The LB service will locate the "upstream" section corresponding to the load balancer **4532** in the Nginx configuration file and identify the "server" entry in it that corresponds to the load balancer's node **271**, and modifies it accordingly.

```
upstream  lb-4532-nodepool {
   …
   server   10.1.1.15:80 weight=4; #node-271
}
```

3.5.4. Remove Node

Verb	URI	Description
DELETE	/loadbalancers/*loadBalancerId*/nodes/*nodeId*	Removes a loadbalancer from an account

Example: Remove Node Request

Request URL: `DELETE /loadbalancers/4532/nodes/271`

HA Proxy

The LB service will locate the "listen" section corresponding to the load balancer **4532** in the HAProxy configuration file and identify the "server" entry in it that corresponds to the load balancer's node **271**, and removes it.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance roundrobin
       server node-271 10.1.1.15:80
```

NetScaler (CLI)

The LB service uses the "rm service" command to remove a service corresponding to node **271**.

```
rm service node-271
```

Nginx

The LB service will locate the "upstream" section corresponding to the load balancer **4532** in the Nginx configuration file and identify the "server" entry in it that corresponds to the load balancer's node **271**, and removes it.

```
upstream  lb-4532-nodepool {
    …
    server    10.1.1.15:80 weight=4; #node-271
}
```

3.6. *Update Load Balancer Attributes*

Verb	URI	Description	Representation
PUT	/loadbalancers/*loadBalancerId*	Update the properties of the load balancer	XML, JSON

Example: Modify LoadBalancer Request: XML

Request URL: `PUT /loadbalancers/4532`

Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<loadBalancer  xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
algorithm="LEAST_CONNECTIONS"  />
```

HA Proxy

The LB service will locate the "listen" section corresponding to the load balancer **4532** in the HAProxy configuration file and change it accordingly.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance leastconn
       server node-271 10.1.1.15:80
```

To change the name of load balancer, you can simply change the name in the listen section to the new name.

NetScaler (CLI)

The LB service uses the "set lb vserver" command to change the attributes of the lb vserver corresponding to the load balancer.

```
set lb vserver lb-4532 -lbmethod LEASTCONNECTION
```

To change the name of an lb vserver, you use the rename command as follows:

```
rename lb vserver lb-4532 newlb-4532
```

The LB service is responsible for maintaining the mapping between the load balancer ID and the new name of the corresponding lb vserver.

Nginx

The LB service will locate the "server" section corresponding to the load balancer **4532** in the Nginx configuration file and change it accordingly.

```
http {
  #lb-4532
  server {
     listen 206.55.130.2:80
     location / {
        proxy_pass http://lb-4532-nodepool;
     }
   }

   #lb-6372
   server {
     …
}
```

By default, the LB algorithm for Nginx is ROUND_ROBIN.

To change the name of load balancer, the LB service can simply edit the comment line above the server section.

Note: The only LB algorithms supported by Nginx are ROUND_ROBIN and SOURCE_IP_HASH. Nginx does not support LEAST_CONNECTIONS algorithm.

3.7. *Usage Reporting*

Verb	URI	Description	Representation
GET	/loadbalancers/*loadBalancerId*/ usage	List current usage	XML, JSON

Example: Get Usage Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<loadBalancerRecords xmlns="http://docs.openstack.org/loadbalancers/api/v1.0" >
        <loadBalancerUsageRecord  incomingTransfer="97301"  outgoingTransfer="241050"
startTime="2010-12-21T12:32:07-06:00"  endTime="2010-12-21T12:36:30-06:00"  />
</loadBalancerRecords>
```

HA Proxy

HA Proxy makes its statistics available from a URL on the HAProxy server itself.

The LB service will enable collecting usage (statistics) on the load balancer by using the appropriate settings on the listen section of HA Proxy's configuration file. An example is shown below to enable statistics to be collected by the LB service from URI **/my_stats** (a username/password can be used to protect this URI).

```
listen lb-4532 206.55.130.2:80
       mode http
       balance leastconn
       server node-271 10.1.1.15:80
       server node-563 10.1.1.14:80
       stats enable
       stats uri /my_stats
       stats refresh 5s
```

The LB service will need to retrieve these stats, parse the data and format it appropriately before returning it to the OpenStack LB client.

NetScaler (CLI)

The LB service uses the "stat lb vserver" command to obtain current statistics corresponding to lb vserver **4532**.

```
stat lb vserver lb-4532
```

NetScaler also supports historical usage statistics. The LB service can use NetScaler's `nsconmsg` tool from the shell on NetScaler to display gathered usage statistics:

```
> /netscaler/nsconmsg -K /var/nslog/newnslog -s time=22Mar2007:20:00 -T
7 -s ConLb=2 -d oldconmsg
```

The output of this command contains a large number of usage counters for each VIP (virtual IP address) and each service. This needs to be filtered by the LB service for the resources that belong to the load balancer or the user account, and then parsed to extract the required usage counters.

An example of the output of this command for one VIP:

```
)VIP(10.102.124.79:0:UP:LEASTCONNS): Hits(1797, 0/sec) Mbps(0.00) Pers(OFF) Err(0) SO(0) LConn_BestIdx: 17
       Pkt(0/sec, 0 bytes) actSvc(1) DefPol(NONE) override(0)
       Conn: Clt(2, 0/sec, OE[2]) Svr(2)
       slimit_SO: (Sothreshhold: 0 [Ex: 0]  Consumed: [Ex: 0 Borrowed: 0 TotActiveConn: 2] Available: 0
```

Nginx

If Nginx is compiled with the *StubStatus* module, you can enable statistics on a server by configuring the `location /nginx_status` directive inside a `server` section which is the URL of a web page on the Nginx web server containing stats about that server.

The LB service will enable collecting usage (statistics) on the load balancer by using the appropriate settings on the listen section of HA Proxy's configuration file. An example is shown below to enable statistics to be collected by the LB service from URI **/my_stats** (a username/password can be used to protect this URI).

```
http {
  #lb-4532
  server {
     listen 206.55.130.2:80
     location / {
        proxy_pass http://lb-4532-nodepool;
     }
     location /nginx_status {
        stub_status on;
        access_log   off;
        allow SOME.IP.ADD.RESS; #only this IP address can access this page
        deny all;
     }
   }
}
```

The LB service will need to retrieve these stats from this URL (`/nginx_status`) periodically, parse the page, extract raw stats and format them appropriately before returning it to the OpenStack LB client.

3.8. *Active Health Monitoring*

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

3.8.1. Connection Monitor

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

- **delay**: The minimum number of seconds to wait before 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.

Example: Retrieve Health Monitor (type CONNECT): XML

URI: /loadbalancers/4532/healthmonitor

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<healthMonitor  xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"  type="CONNECT"
delay="5" timeout="10"  attemptsBeforeDeactivation="3" />
```

HA Proxy

The default health monitoring used in HA Proxy is for HA Proxy to send a simple OPTIONS request at regular intervals to the backend servers.

HA Proxy also supports configuring active monitoring:

The *delay* parameter maps to the HAProxy server's *inter* parameter.

The *timeout* parameter maps to HAProxy's *timeout check* parameter.

The *attemptsBeforeDeactivation* parameter maps to HAProxy's *fall* parameter on the server.

The following shows a configuration example with these parameters set in a listen section corresponding to a load balancer.

```
listen lb-4532 206.55.130.2:80
       mode tcp
       balance leastconn
       timeout check 5000
       server node-271 10.1.1.1:80 check inter 10000 fall 3
       server node-563 1.1.1.2:8080 check inter 10000 fall 3
```

NetScaler (CLI)

NetScaler by default monitors all services of an lb vserver using a default TCP monitor. This requires no configuration.

Users can also configure sevral types of monitors to be used to monitor services. There can be more than one monitor per service, and each service can have different ones.

To enable health monitoring on a load balancer, the LB service needs to create an lb monitor with the required settings configured by the user, and then bind the monitor to all the services of the lbvserver corresponding to the load balancer. Whenever a new node is added to the load balancer, the LB service need to bind the monitor to the service corresponding to the new node.

The *delay* parameter maps to a Netscaler monitor's *interval* parameter.

The *timeout* parameter maps to a Netscaler monitor's *respTimeout* parameter.

The *attemptsBeforeDeactivation* parameter maps to a NetScaler monitor's *retries* parameter.

```
add lb monitor connectmon-4532 tcp -resptimeout 2000 -interval 3000 -retries 3

bind service node-271 connectmon-4532

bind service node-272 connectmon-4532
```

Nginx

Nginx doesn't appear to have default health monitoring. It needs to be configured explicitly.

For HTTP load balancers, if the *ngx_http_healthcheck_module* module is used then health check can be configured as follows:

```
upstream  lb-4532-nodepool {
   healthcheck_enabled;
   healthcheck_delay 3000;
   healthcheck_timeout 2000;
   healthcheck_failcount 3;
   server   10.1.1.1:80; #node-271
   server   10.1.1.2:8080; #node-563
}
```

Where the *delay* parameter maps to the Nginx server's *healthcheck_delay* parameter.

The *timeout* parameter maps to Nginx's healthcheck_*timeout check* parameter.

The *attemptsBeforeDeactivation* parameter maps to Nginx's *healthcheck_failcount* parameter on the server.

If the load balancer is a TCP load balancer, then the configuration of health check will rely on the module *nginx_tcp_proxy_module*:

```
tcp {
        #lb-4532
        server {
            listen 206.55.130.2:3721;
            proxy_pass lb-4532-nodepool;
        }

        upstream lb-4532-nodepool {
            # simple round-robin
            server   10.1.1.1:3721; #71-node1
            server   10.1.1.2:3722; #71-node2
            check interval=3000 fall=5 timeout=1000;

        }
}
```

Where the *delay* parameter maps to the *interval* parameter.

The *timeout* parameter maps to *timeout* parameter.

The *attemptsBeforeDeactivation* parameter maps to *fall* parameter on the server.

3.8.2. HTTP/HTTPS Monitor

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

- **method:** The HTTP method (GET or HEAD) that will be used in the sample request
- **path**: The HTTP path that will be used in the sample request

Example: Update Health Monitor (type HTTPS): XML

URI: /loadbalancers/4532/healthmonitor

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<healthMonitor  xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"  type="HTTPS"
delay="20" timeout="3"  attemptsBeforeDeactivation="5" path="/check"  method="GET"  />
```

HA Proxy

The *method* parameter maps to the HAProxy *httpchk* directive.

The *path* parameter maps to HAProxy's *httpchk* directive.

The following shows a configuration example with these parameters set in a listen section corresponding to a load balancer.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance leastconn
       timeout check 2000
       option httpchk GET /check
       timeout check 3000
       server node-271 10.1.1.1:80 check inter 20000 fall 5
       server node-563 1.1.1.2:8080 check inter 20000 fall 5
```

Updating and deleting the health monitor translates to updates or removal of these directives from the HA Proxy's configuration file.

HA Proxy considers a probe successful if it receives a response with a status code in the 2xx or 3xx range.

Summary of health monitor parameters mappings in HA Proxy

HealthMonitor parameter	HAProxy directive
delay	*inter* argument on *server* directive
timeout	*timeout check* directive
attemptsBeforeDeactivation	*fall* argument on *server* directive
method	Part of *option httpchk* directive
Path	Part of *option httpchk* directive

NetScaler (CLI)

In NetScaler, the method and path parameters can be expressed on an http monitor using the *httpRequest* argument, and you specify the response status codes expected for the probe to be considered successful in the *respCode* argument.

```
add lb monitor httpmon-4532 http -httpRequest "GET /check"-resptimeout 3000
-interval 20000 -retries 5 -respCode 200-399

bind service node-271 httpmon-4532

bind service node-272 httpmon-4532
```

Updating and deleting the health monitor translates to updates or removal (unbinding) of the corresponding NetScaler LB monitors.

Summary of health monitor parameters mappings in NetScaler:

HealthMonitor parameter	NetScaler Monitor argument
delay	*interval*
timeout	*respTimeout*
attemptsBeforeDeactivation	*retries*
method	Part of *httpRequest*
Path	Part of *httpRequest*

Nginx

For using HTTP health monitors, Nginx will rely on *ngx_http_healthcheck_module* directives.

```
upstream  lb-4532-nodepool {
   healthcheck_enabled;
   healthcheck_delay 3000;
   healthcheck_timeout 2000;
   healthcheck_failcount 3;
   # Important: use HTTP/1.0
   healthcheck_send "GET /check HTTP/1.0" 'Host: www.mysite.com';
   healthcheck_expected 'I_AM_ALIVE';
   server   10.1.1.1:80; #node-271
   server   10.1.1.2:8080; #node-563
}
```

Where the *method* parameter maps to a part of *healthcheck_send* directive.

The *path* parameter maps to a part *healthcheck_expected* directive.

Note: Nginx's *ngx_http_healthcheck_module* considers a health probe to be successful if the HTTP response status code is 200, and the body matches the *health_expected* value. The status codes expected are not configurable.

Updating and deleting the health monitor translates to updates or removal of these directives from the Nginx's configuration file.

Summary of health monitor parameters mappings in Nginx:

HealthMonitor parameter	Nginx directive
delay	*healthcheck_delay* directive
timeout	*healthcheck_timeout* directive
attemptsBeforeDeactivation	*healthcheck_failcount* directive
method	Part of *healthcheck_send* directive
Path	Part of *healthcheck_send* directive

3.9. *Session Persistence*

Verb	URI	Description	Representation
GET	/loadbalancers/*loadBalancerId*/sessionpersistence	List session persistence configuration.	XML, JSON

PUT	/loadbalancers/*loadBalancerId*/sessionpersistence	Enable session persistence.	XML, JSON
DELETE	/loadbalancers/*loadBalancerId*/sessionpersistence	Disable session persistence.	

Example: Enable session persistence: XML

URI: /loadbalancers/4532/sessionpersistence

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<sessionPersistence  xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
persistenceType="HTTP_COOKIE" />
```

HA Proxy

The following shows a configuration example of enabling cookie persistence on a load balancer using the *cookie* directive (with the *insert* value) in the listen section corresponding to the load balancer.

This configuration inserts a cookie called "cookie-4532" into responses, and removes this cookie from requests. The cookie will have the value "server1" if the LB algorithm chooses node-271 for a new user, and "server2" if node-563 is chosen. Subsequent requests from the same user will go to the same node.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance leastconn
       cookie cookie-4532  insert indirect
       server node-271 10.1.1.1:80 cookie server1
       server node-563 10.1.1.2:8080 cookie server2
```

To delete session persistence on a load balancer, you remove the *cookie* directive from the corresponding listen section in the configuration file and restart HAProxy.

In addition to HTTP Cookie persistence, HAProxy supports another persistence type: RDP cookies.

NetScaler (CLI)

To enable HTTP_COOKIE persistence on a load balancer in Netscaler, you locate the corresponding lbvserver and set the *persistenceType* parameter to COOKIEINSERT.

```
set lb vserver lb-4532 –persistenceType INSERTCOOKIE
```

To disable session persistence, you set the same parameter to the NONE value.

```
set lb vserver lb-4532 –persistenceType NONE
```

Besides "HTTP COOKIE" persistence, NetScaler supports the following persistence types:

- SOURCE IP
- SSL SESSION ID
- RULE (any NetScaler expression)
- URLPASSIVE
- DESTINATION IP
- SOURCE IP DESTINATION IP
- CALL ID
- RTSP SID
- CUSTOM SERVER ID

Nginx

Note: Nginx's *Upstream* module supports only **source IP hash** session persistence. This is configured in the "upstream" section corresponding to the load balancer.

```
upstream  lb-4532-nodepool {
   ip_hash;
   server   10.1.1.1:80; #node-271
   server   10.1.1.2:8080; #node-563
}
```

There is an nginx module called *nginx_sticky_module* released in June 2010, that provides support for HTTP cookies, but it is not an official nginx module, doesn't seem to be stable (rc release) and the open source project for it doesn't show much activity since release.

```
upstream  lb-4532-nodepool {
   sticky;
   server   10.1.1.1:80; #node-271
   server   10.1.1.2:8080; #node-563
}
```

3.10. *Connection Logging*

3.10.1. Enable or Disable Connection Logging on a Load Balancer

Verb	URI	Description	Representation
PUT	/loadbalancers/*loadBalancerId*/connectionlogging	Enable or Disable connection logging	XML, JSON

Example: Enable connection logging: XML

URI: /loadbalancers/4532/connectionlogging

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<connectionLogging  xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"  enabled="true"/>
```

HA Proxy

On HAProxy, you use the option *httplog* to enable HTTP-style (CLF format) logging in the listen section corresponding to the load balancer.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance leastconn
       option httplog clf
        server node-271 10.1.1.1:80
       server node-563 10.1.1.2:8080
```

NetScaler (CLI)

For HTTP logging, you need to enable WebLogging on NetScaler (W3C format) by editing the `log.conf` file on NetScaler and adding the following directives:

```
begin default
logFormat W3C
logInterval Daily
logFilenameFormat Ex%{%y%m%d}t.log
end default
```

Then start the WebLogging daemon on NetScaler as follows:

```
> nswl –start –f <location of log.conf file>
```

To disable HTTP logging, stop the WebLogging daemon.

> **Note**: At the current time, HTTP (Web) logging configuration in NetScaler is global and cannot be enabled/disabled per lbvserver.

For logging TCP connections, NetScaler makes use of audit policies. These audit policies can then be bound to the lbvserver corresponding to the load-balancer to enable TCP logging for the corresponding load balancer.

The following example shows creating an audit policy to log TCP events to syslog, and then binding this policy to the lbvserver corresponding to the load balancer.

```
add audit syslogAction tcplogAction-4532 127.0.0.1 –logLevel INFORMATION –
tcp ALL
add audit syslogpolicy tcplogPolicy-4532 ns_true tcplogAction-4532
bind lb vserver lb-4532 –policy tcplogPolicy-4532 –priority 10
```

On Nginx, the LB service would use the directive *log_format* to create an Apache-style format for a log entry, then in the server section, the *access_log* directive is used to specify the name of the log file to be used for this server section and the name of the log format specified earlier.

```
http {
  log_format   mylogformat '$remote_addr - $remote_user [$time_local]  $status '
    '"$request" $body_bytes_sent "$http_referer" '
    '"$http_user_agent" "$http_x_forwarded_for"';


  #lb-4532
  server {
     listen 206.55.130.2:80
     access_log logs/lb-4532.log mylogformat;
     location / {
        proxy_pass http://lb-4532-nodepool;
     }
   }
}
```

3.11. *Connection Throttling*

3.11.1. Update Connection Throttling configuration

Verb	URI	Description	Representation
PUT	/loadbalancers/*loadBalancerId*/connectionthrottle	Update configuration of connection throttling	XML, JSON

Example: Update configuration of connection throttling: XML

URI: /loadbalancers/4532/connectionthrottle

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<connectionThrottle  xmlns="http://docs.openstack.org/loadbalancers/api/v1.0"
maxConnectionRate="50" rateInterval="1" />
```

HA Proxy

On HA Proxy, the LB service would use various directives to enable connection throttling per source IP address.

The following example shows connection throttling allowing a maximum of 50 requests every second per source IP address. Beyond that, connections are refused to prevent abuse.

```
listen lb-4532 206.55.130.2:80
       mode http
       balance leastconn
       # declare table to store rate (connections/s) per source IP
       stick-table type ip size 1m expire 5m store http_req_rate(1s)
       # enable tracking of connections on VIP from each source IP.
       tcp-request content track-sc1 src
       #boolean expression: true if connection rate is above limit (50).
       acl above_rate sc1_http_req_rate gt 50
       #reject connections if the expression above_rate is true
       tcp-request content reject if above_rate
       server node-271 10.1.1.1:80
       server node-563 10.1.1.2:8080
```

In addition to requests/second, HAProxy can track connections/second. The action to be taken when rate is exceeded is configurable.

HA Proxy can also rate limit the number of connections accepted on a VIP per second (considering all clients) using the "`rate-limit sessions <number>`" directive. It can also limit the overall total number of connections on a VIP using the "`maxconn <directive>`" directive.

Summary of Connection Throttling mapping to HAProxy

Connection Throttling	HA Proxy directives
maxRequestRate	Track client source IP by using directive "`tcp-request content track-sc1 src`". Use `acl` directive to define expression of rate together with `tcp-request content reject` to drop connections that exceed that rate.
rateInterval	Define a `stick-table` and specify rateInterval as a `http_req_rate` parameter

NetScaler (CLI)

On Netscaler, the LB service would make use of the rate-limiting feature to create a *limitSelector* to track source IP addresses and a *limitIdentifier* where the required rate limits can be specified. The LB service would then use this limitIdentifier in a *responder policy* that drops connections from a client when the rate is exceeded from a certain source IP. Finally the policy is bound to the lbvserver corresponding to the load balancer.

```
add ns limitSelector client_src_ip CLIENT.IP.SRC
add ns limitIdentifier above_rate -mode REQUEST_RATE -selectorName
client_src_ip -threshold 50 -timeSlice 1000
add responder policy rate_policy SYS.CHECK_LIMIT("above_rate") DROP
bind lb vserver lb-4532 -policy rate_policy -priority 10
```

NetScaler *limitIdentifier* can also be used to track max concurrent connections (although not per time interval) as well as requests/second as we used it in the example above.

Also, what is selected for tracking is not limited to client source IPs. It can be any part or characteristic of the client request, including content of request, destination IP address, time of day, etc. If a connection or a request matches the criteria, it is counted for the purpose of rate limiting. The action to take when the rate limit is exceeded can be based on any of the NetScaler policies (cache, respond, content switch, rewrite, filter, etc.).

Summary of Connection Throttling mapping to NetScaler

Connection Throttling	NetScaler rate-limiting
maxRequestRate	Define a limitSelector that tracks client source IP. Then define a limiteIdentifier that uses the selector, and specify *mode* as REQUEST_RATE and specify in the *threshold* the number of requests to admit per second.
rateInterval	*timeSlice* which is the number of milliseconds (in mulitples of 10) where the rate of requests is evaluated.

Nginx

The LB service would use Nginx's HttpLimitReqModule module directive *limit_req_zone* to define a zone with the required rate limits. It then uses the

limit_req in a server section to specify the zone (rate limits) to be used for that server.

The following example shows connection throttling allowing a maximum of 50 requests every second per source IP address (denoted by variable `$binary_remote_addr` in zone definition). Beyond that, connections are refused to prevent abuse.

```
http {
  limit_req_zone  $binary_remote_addr  zone=firstzone:10m   rate=50r/s;


  #lb-4532
  server {
     listen 206.55.130.2:80
     location / {
        limit_req   zone=firstzone;
        proxy_pass http://lb-4532-nodepool;
     }
   }
}
```

Nginx *limit_req* directive also supports specifying number of burst requests, in which case, requests that exceed the rate but are still under the burst limit will be delayed rather than refused.

Summary of Connection Throttling mapping to Nginx:

Connection Throttling	Nginx directives
maxRequestRate rateInterval	Use the *limit_req_zone* to define the maxRequestRate per rateInterval, you want to admit from each source IP address. Then use a *limit_req* directive in the server section corresponding to the load balancer.

3.12. *Load Balancing Protocols*

Verb	URI	Description	Representation

GET	/loadbalancers/protocols	List all supported load balancing protocols	

HA Proxy

HA Proxy supports HTTP and TCP protocols. The LB service would specify the load-balanced protocol in HA Proxy using the *mode* directive which can take the values `http` or `tcp`.

For HTTPS (and SSL in general), as well as other non-HTTP protocols (SMTP, LDAP, etc.), the "tcp" mode is used.

```
listen lb-4532 206.55.130.2:80
       mode tcp
       balance roundrobin
       server node-271 10.1.1.1:80
       server node-563 10.1.1.2:8080
```

NetScaler (CLI)

NetScaler also supports HTTP and TCP protocols.

The LB service would specify the protocol when creating the lb vserver corresponding to the load balancer. NetScaler supports the following protocols when creating an lb vserver:

1. HTTP
2. TCP
3. UDP
4. SSL
5. SSL_BRIDGE
6. SSL_TCP
7. FTP
8. MYSQL
9. MSSQL
10. RDP
11. RTSP
12. RADIUS
13. SIP_UDP
14. NNTP
15. DNS

16.DNS_TCP
17.UDP
18.ANY

```
add lb vserver a-new-loadbalancer <protocol> 206.55.130.2 80
```

Nginx

HTTP loadbalancers are configured inside an "http {}" section as follows:

```
http {

  #lb-4532
  server {
     listen 206.55.130.2:80
     server_name .*;
     location / {
        proxy_pass http://lb-4532-nodepool;
     }
   }
}
```

```
http {

  #lb-4532
  server {
      listen 206.55.130.2:443;
      server_name .*;
      ssl on;
      ssl_certificate /etc/nginx/ssl/mysite.com.pem;
      ssl_certificate_key /etc/nginx/ssl/mysite.com.pem;
      ssl_session_timeout 5m;
      proxy_pass http://lb-4532-nodepool;
   }
}
```

If you want SSL pass-through instead (prevent Nginx from performing SSL offloading), then you should be able to achieve this by using the TCP module (see later), and treating HTTPS connections simply as TCP connections that need to be load balanced.

Nginx also supports load balancers for mail protocols (SMTP, IMAP, POP3). These are configured inside a "mail {}" section. For example, below we configure a POP3 load balancer:

```
mail {

  server_name .*
  auth_http  127.0.0.1:80/auth;
  pop3_auth plain;


  #lb-4571
  server {
     listen 206.55.130.2:110
     protocol pop3;
     proxy on;
   }
}
```

The *auth_http* directive specifies the authentication server that Nginx uses for a new user. The auth server will authenticate the user and tell Nginx to which POP3 server to forward the request, so there is no need to configure a set of backend nodes (an *upstream* section like for http) in this case. The configuration of backend nodes would be done on the authentication server used by Nginx.

You can also enable SSL on the mail protocols to support POP3S, IMAPS protcols by using the ssl directives of the *MailSslModule* of Nginx.

Using the *nginx_tcp_proxy_module* module, the TCP protocol support can be added to Nginx. The following example shows configuration for a TCP load balancer:

```
tcp {
        #lb-4549
        server {
            listen 206.55.130.2:3721;
            proxy_pass lb-4549-nodepool;
        }


        upstream lb-4549-nodepool {
            # simple round-robin
            server   10.1.1.1:3721; #71-node1
            server   10.1.1.2:3722; #71-node2
        }

}
```

3.13. *Load Balancing Algorithms*

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

HA Proxy

HA Proxy supports ROUND_ROBIN and LEAST_CONNECTIONS algorithms. Below is the full list of supported Load Balancing Algorithms:

1. roundrobin- Round Robin
2. leastconn – Least Connections
3. source – Hash of the Source IP address
4. uri – Hash of the URI (path)
5. url_param - Based on value of URL parameter
6. hdr(name) – Based on the value of a request header
7. RDP-cookie(name) - Based on the cookie found in Microsoft RDP protocol

```
listen lb-4532 206.55.130.2:80
       mode http
       balance url_param userid
       server node-271 10.1.1.1:80
       server node-563 10.1.1.2:8080
```

NetScaler (CLI)

OpenStack LB algorithms map in NetScaler as follows:

Load Balancer Algorithm	LB vserver method
ROUND_ROBIN	roundrobin
LEAST_CONNECTIONS	leastconnection

The full list of Load Balancing Algorithms supported by NetScaler:

1. ROUND ROBIN
2. LEAST CONNECTIONS
3. LEAST RESPONSE TIME

4. URL HASH
5. DOMAIN HASH
6. DESTINATION IP HASH
7. SOURCE IP HASH
8. LEAST BANDWIDTH
9. LEAST PACKETS
10. SOURCE IP DESTINATION IP HASH
11. SOURCE IP SOURCE PORT HASH
12. CALL ID HASH
13. LRTM (LEAST CONNECTIONS taking response time into account)
14. CUSTOM LOAD

Below is the command for creating an lbvserver that uses round-robin as the load balancing algorithm

```
add lb vserver a-new-loadbalancer http 206.55.130.2 80 -lbmethod roundrobin
```

Nginx

Nginx supports only ROUND_ROBIN (the default) and SOURCE_IP_HASH algorithms.

There is no directive for configuring ROUND_ROBIN. To configure SOURCE_IP_HASH instead of ROUND_ROBIN, a directive *ip_hash* in the *upstream* section of the configuration file is added: See Session Persistence section in this document for an example configuration.

Note: Nginx doesn't support the LEAST_CONNECTIONS algorithm.

3.14. *Load Balancer Status*

Table: 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
ERROR	The system encountered an error when attempting to configure the load balancer

HA Proxy

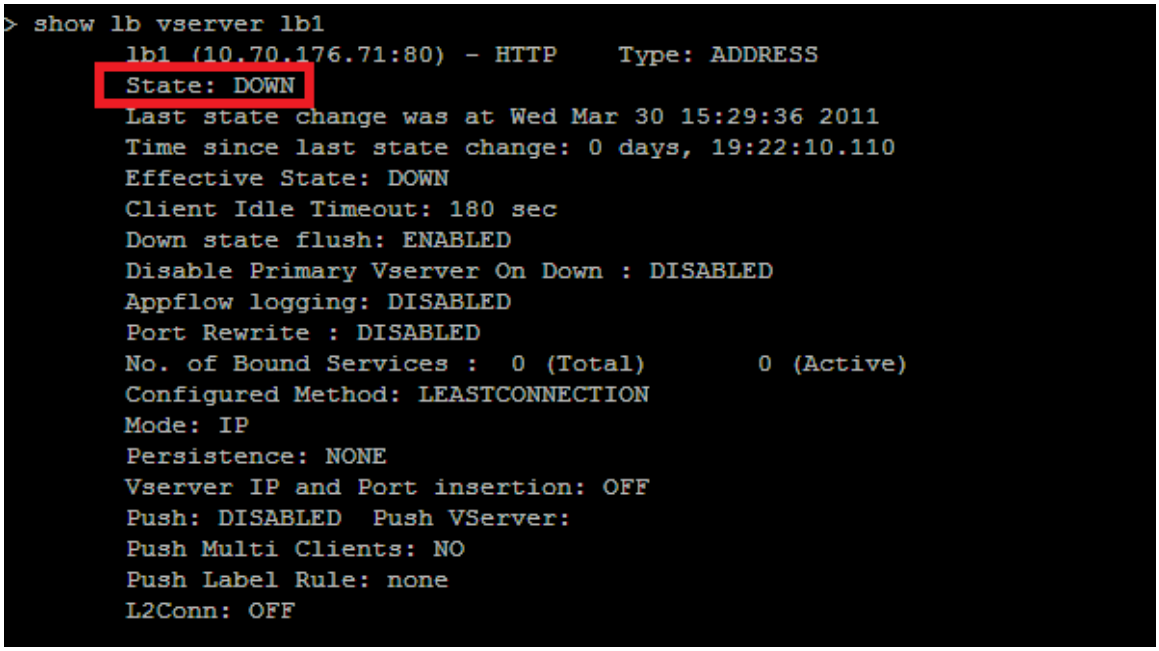
HAProxy is a Linux process, and therefore the LB service has to monitor its status (if the process is working properly).

Typical deployments on Linux use `keepalived` or `heartbeat` daemon to restart the HAProxy process if it stops.

To disable a load balancer, the LB service would need to remove the corresponding "listen" section from HAProxy config file, and restart the process with the new config file. There is no admin command to ask HA Proxy to stop listening on specific IP addresses.

NetScaler (CLI)

On NetScaler to find out the current status of the lb vserver corresponding to the load balancer, you use the command "show lb vserver". An output of the execution of this command is shown below:



The status of the lb vserver can be either UP, DOWN or OUT OF SERVICE (when explicitly disabled).

The lb vserver status can be used by the LB service to deduce the status of the corresponding load balancer.

Load Balancer status	Lb vserver state
ACTIVE	UP
BUILD	-
PENDING_UPDATE	-
PENDING_DELETE	-
SUSPENDED	OUT OF SERVICE
ERROR	-

Nginx

Nginx is a Linux process, and therefore the LB service has to monitor its status (if the process is working properly).

Typical deployments on Linux use `keepalived` or `heartbeat` daemon to restart the Nginx process if it stops.

To disable a load balancer, the LB service would need to remove the corresponding "server" section from Nginx config file, and restart the Nginx master process with the new config file. There is no admin command to ask Nginx to stop listening on specific IP addresses.

3.15. *Node Condition*

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

Table: 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.

HA Proxy

On HA Proxy you change the condition of a node by enabling/disabling the corresponding server using a "stats admin socket" to issue the following admin commands to HAProxy on a Unix socket.

```
enable server node-271
```

```
disable server node-563
```

Where "DISABLE" mode on HA Proxy is really a "draining" mode: it doesn't kill existing connections being serviced by that node, it simply stops the node for receiving new connections.

NetScaler (CLI)

On NetScaler you change the condition of a node by enabling/disabling the corresponding service.

```
enable service node-271
```

```
disable service node-563
```

Whether existing connections are killed when the service is disabled, depends on the "downstate flush" setting of the service. If this setting is enabled, then connections are forcibly closed when the service is disabled. This setting is enabled by default.

You can also disable the service with a delay during which requests for existing sessions (when session persistence is used) are still sent to the service, but requests from new clients are not. After the delay, no new requests are sent to the service (regardless of persistence).

Below is an example of delaying the service disabling by 5mn.

```
disable service node-563 300
```

In the latest NetScaler release (9.3), you can also disable services gracefully: The service will stay UP as long as there are existing connections established. No new connections however are accepted. This is useful for TCP protocols with long-lived connections (e.g. SMTP, etc.).

```
disable service node-563 -graceFul YES
```

The LB service can discover the current status of a node by finding out the status of its corresponding service using the command "`show service <servicename>`". The output is similar to the one shown for the lbvserver above.

Nginx

On Nginx you change the condition of a node by marking it as "down" in the configuration file.

In the following example, we disable the first node (10.1.1.1) of the load balancer.

```
http {
        #lb-4557
        server {
            listen 206.55.130.2:80;
            proxy_pass http://lb-4557-nodepool;
        }


        upstream lb-4557-nodepool {
            # simple round-robin
            server   10.1.1.1:80 down; #235-node1
            server   10.1.1.2:80; #623-node2
        }

}
```