



Barracuda Load Balancer API Guide

Version 1.0

Barracuda Networks, Inc.
3175 S. Winchester Blvd
Campbell, CA 95008
<http://www.barracuda.com>

Contents

Introduction	4
Requirements	4
Available APIs	4
Getting Started.....	5
Configure the Barracuda Load Balancer to Accept API Calls.....	5
Install Perl and XML-RPC on the Control System	5
Linux systems	5
Windows systems	5
Mac OS-X systems	5
Using the API.....	6
Creating the XML-RPC Object	6
Return Codes	6
Successful Results.....	7
Failure Results	7
API Descriptions	8
service.add	8
Example	8
Result	8
service.delete	9
Example	9
Result	9
service.show.....	10
Example	10
Result	10
server.add	11
Example	11
Result	11
server.delete.....	12
Example	12
Result	12
server.change_state.....	13
Example	13
Result	13
Sample Program – Multiple Functions.....	14
sample.pl	15
Sample Program – Changing Real Server Status to Maintenance Mode	16
maint.pl.....	17

Introduction

Use the Barracuda Load Balancer API to automate a variety of tasks including:

- Add or delete a Service
- Add or delete a Real Server
- Show the state and configuration data for a Service or Real Server
- Enable or disable a Real Server or put a Real Server into maintenance mode

Note: The interface is implemented using a set of XML-RPC requests and parameters that specify the inputs. The samples included in this document are written in Perl, but you can call the API from any programming language that supports XML-RPC calls.

For a complete example Perl program, see *Sample Program – Multiple Functions* on page 14.

Requirements

- Barracuda Load Balancer running firmware version 3.6.1.009 or higher
- Barracuda Load Balancer model 440 or above is required

Available APIs

The following table describes the APIs available for the Barracuda Load Balancer:

API	Description
<code>service.add</code>	Add a Service
<code>service.delete</code>	Delete a Service
<code>server.add</code>	Add a Real Server to a Service
<code>service.delete</code>	Delete a Real Server from a Service
<code>service.show</code>	Display configuration and state information for a Service and/or a Real Server
<code>server_change_state</code>	Change the state of a Real Server to disable / enable / maintenance mode

Getting Started

Configure the Barracuda Load Balancer to Accept API Calls

Before running any program, configure the Barracuda Load Balancer to accept calls from the external system.

On the Barracuda Load Balancer,

1. Navigate to the **BASIC > Administration** page.
2. In the **Allowed API IP/Range** table, enter the IP address of the system from where the API call is to be made, and click **Add**.
3. In the **API Password** field, enter the API password.

Caution: Attempts to use the API from an IP address that is not in the table are denied.

Install Perl and XML-RPC on the Control System

Because the APIs are all XML-RPC methods, XML-RPC must be installed on the control system. Use the following steps to install perl and XML-RPC on the associated system.

Linux systems

1. Install perl.
2. To start the CPAN shell, type:
`perl -MCPAN -we 'shell'`
3. From the CPAN shell, type:
`install XML::RPC`

Windows systems

1. Install perl (for example, ActivePerl or Strawberry Perl).
2. From the command prompt, type:
`cpan.bat`
3. Once in CPAN, type:
`install XML::RPC`

Mac OS-X systems

1. Install perl.
2. Install CPAN.
3. Set FTP into passive mode so that CPAN can use it.
4. In your bash shell, or your `~/.bash_login` file, add the following line:
`export FTP_PASSIVE=1`
5. Log into CPAN, and use it to install XML-RPC:
`cpan[2]> get XML::RPC`
`cpan[2]> install XML::RPC`

Using the API

Creating the XML-RPC Object

As shown in the sample code below, the first step is to create an XML-RPC object that includes the WAN IP address of the Barracuda Load Balancer, password, and timeout value. The request is an HTTP POST to /cgi-mod/api.cgi:

```
http://<Barracuda Load Balancer WAN IP address: port>/
cgi-mod/<api>?password=API password
```

For improved security, using https (default port 443) for sending the password is recommended:

```
https://<Barracuda Load Balancer WAN IP address>/cgi-
mod/<api>?password=API password
```

```
#!/usr/bin/perl
use strict;
use warnings;

use XML::RPC;
use LWP::UserAgent;
use Data::Dumper;

my $ua = LWP::UserAgent->new ;

$ua->timeout( 60 ) ;

# Create the XML::RPC object
my $xmlrpc = XML::RPC->new (
'https://10.5.126.63/cgi-mod/api.cgi?password=admin',
'lwplib_useragent' => $ua );
```

Return Codes

If there is an error, the XML-RPC `new` call returns a hash reference containing two entries: a fault code and a descriptive string.

If the password given is wrong, the hash returned is:

faultCode	faultStr
403	Access denied: Configuration administration password mismatch.

If the caller IP address is not within the allowed API/IP range, the hash returned is:

faultCode	faultStr
403	Access denied: Your machine does not have access to administer configuration.

For all of the API calls described in the rest of this document except for the `service.show` API, the hash returned is a value and a message:

```
$result = $xmlrpc->call( ... );
$val = $$result{val};
$msg = $$result{msg};
```

The possible value and message pairs returned are listed in the following section.

Successful Results

The output of a successful call is a hash with a value and a message. The message varies depending on the call made:

val	msg
200	Service successfully added OR Server successfully added OR Server successfully deleted OR Service successfully deleted OR Server mode successfully changed

Failure Results

The output of a failed call is a hash with a value and a message. The `msg` column in this table shows some of the returned messages:

val	msg
700	Invalid parameter in configuration
702	ERROR: Service already exists OR ERROR: Server already exists OR ERROR: (passed IP Address) does not exist OR ERROR: Service does not exist
704	ERROR: (passed value) is an invalid action
707	This operation is not permitted until this Barracuda Load Balancer has been activated OR Error in the parameters passed, e.g., ERROR: Illegal value (passed value) for Real Server: must be in IP address notation.

API Descriptions

service.add

This method adds a Service to the configuration database. The VIP address and port combination must be unique.

Parameter	Description
name	Service name
vip	Set this to the Virtual IP address of the Service
protocol	The protocol used by the Service, either TCP or UDP
port	The port used by the Service
type	<p>The Service type. Supported values:</p> <ul style="list-style-type: none"> L4 with protocol UDP adds a Layer 4 – UDP Service L4 with protocol TCP adds a Layer 4 – TCP Service L7 with protocol TCP adds a Layer 7 – HTTP Service L7TCP with protocol TCP adds a TCP Proxy Service L7UDP with protocol UDP adds a UDP Proxy Service FTP with protocol TCP adds a Layer 7 – FTP Service <p>Other combinations are not valid. Adding SSL offloaded Services (e.g., Layer 7 – HTTPS) is not supported.</p>

Example

```
my $result;

#Add a Service
$result = $xmlrpc->call('service.add'
{name=>'xml_rpc_test',vip=>'192.168.132.214',protocol=>'TCP',port=>'21',
type=>'L4'});

print Dumper ($result);
```

Result

The output of a successful call is a hash with a value and a message:

```
val          msg
200          Service successfully added
```

```
$VAR1 = {
    'msg' => 'Service successfully added',
    'val' => '200'
};
```

service.delete

This method deletes an existing Service from the configuration database.

Parameter	Description
vip	Set this to the Virtual IP address of the Service
protocol	The protocol used by the Service, either TCP or UDP
port	The port used by the Service

Example

```
my $result;

$result = $xmlrpc->call('service.delete'
{vip=>'192.168.132.214',port=>'21', protocol=>'TCP'});

print Dumper ($result);
```

Result

The output of a successful call is a hash with a value and a message:

```
val      msg
200      Service successfully deleted
```

```
$VAR1 = {
    'msg' => 'Service successfully deleted',
    'val' => '200'
};
```

service.show

This method displays configuration and state information for a Service and/or a Real Server. If no parameters are entered to identify one Service or Real Server, information about all existing Services and Real Servers is returned.

Parameter	Description
vip	Optional. Set this to the Virtual IP address of the Service that this Real Server is to be associated with, followed by a colon, followed by TCP or UDP, i.e., VIP address: protocol
ip	Optional. The IP address of the Real Server
port	Optional. The port used by the Real Server
show	A list of what is to be shown, separated by '/'. For example, to show the status of a Service and the state of a Real Server, use <code>status/state</code>

Example

```
my $result;

#Show Service status and Real Server state
my $service_show = 'status/state' ;
$result = $xmlrpc->call('service.show',
{vip=>'192.168.132.214:21:TCP', port=>'21', ip=>'15.15.15.11',
show=>$service_show});

print Dumper ($result);
```

Result

The output of a successful call is a hash for each Service with a set of variables:

Variable	Description
proto	TCP or UDP
ip	VIP address of the Service
status	Service status – down or up
name	Service name
port	Service port

and a hash for Real Server with a set of variables:

Variable	Description
ip	Address of the Real Server
status	Real Server status – disabled or enabled
port	Real Server port
state	Real Server state – maintenance, disabled, or enabled

server.add

This method adds a Real Server to the configuration database. It must be added to an existing Service.

Parameter	Description
vip	Set this to the Virtual IP address of the Service that this Real Server is to be associated with, followed by a colon, followed by TCP or UDP, i.e., VIP address: protocol
ip	The IP address of the Real Server
port	The port used by the Real Server

Example

```
my $result;

#Add a Real Server
$result = $xmlrpc->call('server.add',
{vip=>'192.168.132.214:21:TCP',ip => '15.15.15.11',port => '21'});

print Dumper ($result);
```

Result

The output of a successful call is a hash with a value and a message:

```
val          msg
200          Server successfully added
```

```
$VAR1 = {
    'msg' => 'Server successfully added',
    'val' => '200'
};
```

server.delete

This method removes an existing Real Server from an associated Service.

Parameter	Description
vip	Set this to the Virtual IP address of the Service that this Real Server is associated with, followed by a colon, followed by TCP or UDP, i.e., VIP address: protocol
ip	The IP address of the Real Server
port	The port used by the Real Server

Example

```
my $result;

#Delete the Real Server
$result = $xmlrpc->call('server.delete',
{vip=>'192.168.132.214:21:TCP',ip => '15.15.15.11',port => '21'});

print Dumper ($result);
```

Result

The output of a successful call is a hash with a value and a message:

```
val          msg
200           Service successfully deleted
```

```
$VAR1 = {
    'msg' => 'Server successfully deleted',
    'val' => '200'
};
```

server.change_state

This method changes the state of a Real Server to disabled, maintenance, or enabled. Changing it to disabled terminates all connections immediately. When placed in maintenance mode, the Real Server keeps existing connections but does not accept any new ones. Maintenance can be performed when all active connections are closed.

Parameter	Description
vip	Set this to the Virtual IP address of the Service that this Real Server is associated with, followed by a colon, followed by TCP or UDP, i.e., VIP address: protocol
ip	The IP address of the Real Server
port	The port used by the Real Server
action	Set to one of the following strings: disable / enable / maintenance

Example

```
my $result;

#Change Real Server state to (disabled / enabled / maintenance mode)

$result = $xmlrpc->call('server.change_state', {vip=>'192.168.132.214:21:TCP',
port=>'21', ip => '15.15.15.11', action=>'disable'}); print Dumper ($result);

print Dumper ($result);
```

Result

The output of a successful call is a hash with a value and a message:

```
val          msg
200          Server mode successfully changed
```

```
$VAR1 = {
    'msg' => 'Server mode successfully changed',
    'val' => '200'
};
```

Sample Program – Multiple Functions

This program uses most of the API calls described in the section *API Descriptions* beginning on page 8. It performs the following functions:

- Adds a Service and a Real Server;
- Shows the Service and Real Server state before and after setting the state of the Real Server to maintenance mode;
- Deletes the Real Server and the Service.

Note: You can copy and paste this code into a file and save it as `sample.pl`. Edit the IP addresses – 10.5.126.63 should be the WAN IP address of the Barracuda Load Balancer, 192.168.132.214 is the VIP address of a Service, and 15.15.15.11 is the IP address of a Real Server. Change the password to the API password for the Barracuda Load Balancer.

To run `sample.pl`, at a command prompt type:

```
perl sample.pl
```

sample.pl

```
#!/usr/bin/perl
use strict;
use warnings;

use XML::RPC;
use LWP::UserAgent;
use Data::Dumper;

my $ua = LWP::UserAgent->new ;

$ua->timeout( 60 ) ;

# Create the XML::RPC object
my $xmlrpc = XML::RPC->new (
    'https://10.5.126.63/cgi-mod/api.cgi?password=admin',
    'lwp_useragent' => $ua );
my $result;

#Add a Service
$result = $xmlrpc->call('service.add',
    {name=>'xml rpc test',
    vip=>'192.168.132.214',
    protocol=>'TCP',
    port=>'21',
    type=>'L4'});
print Dumper ($result);

#Add a Real Server
$result = $xmlrpc->call('server.add',
    {vip=>'192.168.132.214:21:TCP',ip => '15.15.15.11',port => '21'});
print Dumper ($result);

my $service show = 'status/state' ;
#Show all Services and Real Servers state
$result = $xmlrpc->call('service.show', {show=> $service show});

print Dumper ($result);

#Change Real Server status to maintenance mode (could be one of disable/enable/maintenance)
$result = $xmlrpc->call('server.change state',
    {vip=>'192.168.132.214:21:TCP',
    port=>'21',
    ip => '15.15.15.11',
    action=>'maintenance'});
print Dumper ($result);

#Show Service status and Real Server state

$result = $xmlrpc->call('service.show',
    {vip=>'192.168.132.214:21:TCP',
    port=> '21', ip=> '15.15.15.11',
    show=> $service show});
print Dumper ($result);

#Delete the Real Server
$result = $xmlrpc->call('server.delete',
    {vip=>'192.168.132.214:21:TCP',
    ip => '15.15.15.11',
    port => '21'});
print Dumper ($result);

#Delete the Service
$result = $xmlrpc->call('service.delete',
    {vip=>'192.168.132.214',
    port=>'21',
    protocol=>'TCP'});
print Dumper ($result);
```

Sample Program – Changing Real Server Status to Maintenance Mode

This program changes the status for a list of Real Servers to maintenance mode. It performs the following functions:

- For each IP address in a list, it sets the status of the Real Server to maintenance. The Barracuda Load Balancer will not direct any new client connections or requests to a Real Server in maintenance mode;
- It displays the Real Server state to confirm that the state has changed.

Notes: Edit the IP addresses – 10.5.126.63 should be the WAN IP address of your Barracuda Load Balancer, 192.168.132.214 is the VIP address of an existing Service, and 15.15.15.11 and 15.15.15.12 are the IP addresses of the Real Servers to be put into maintenance mode. You can put more Real Servers in the `$server_ips` list, just be sure to separate them by a comma (.). Change the password to the API password for the Barracuda Load Balancer.

You can also modify this program to disable the Real Servers or make them active again by changing the value of the `$action` parameter to `disable` or `enable`.

To run `maint.pl`, at a command prompt type:

```
perl maint.pl
```

maint.pl

```
#!/usr/bin/perl

use strict;
use warnings;

use XML::RPC;
use LWP::UserAgent;

# Service parameters
my $service_ip = '192.168.132.214' ;
my $service_protocol = 'TCP' ;
my $service_port = '80' ;

# Server parameters
my $server_ips = '15.15.15.11,15.15.15.12' ;
my $server_port = '80' ;

my $method_change = "server.change_state" ;
my $action = 'maintenance' ; #disable/enable/maintenance
my $service_id = "$service_ip:$service_port:$service_protocol" ;
my @server_ip = split(",", $server_ips);
my $service_show = 'status/state' ;
my $result;

# Create the XML::RPC object
my $ua = LWP::UserAgent->new ;
$ua->timeout( 60 );
my $xmlrpc = XML::RPC->new (
    'https://10.5.126.63/cgi-mod/api.cgi?password=admin',
    'lwp_useragent' => $ua );

#Change Real Server status for the IP addresses in the list
#to maintenance mode

my $disable_parameters = {
    vip => $service_id,
    port => $server_port,
    action => $action };

foreach my $ip (@server_ip){
    $disable_parameters->{'ip'} = $ip;
    print "Setting status for server $ip to $action.\n";
    $result = $xmlrpc->call($method_change, $disable_parameters );
    print "$result->{'msg'}\n";

    #Show the Real Server state now
    $result = $xmlrpc->call( 'service.show',
        { vip => $service_id,
          port => $server_port,
          ip => $ip,
          show => $service_show
        }
    );

    my $instance_id = "$ip" . ":" . "$server_port";
    print
    "Server status: $result->{${service_id}}{${instance_id}}{'state'}\n";
}

```