Stingray™ Traffic Manager
REST API Guide

Version 9.2

May 2013

riverbed®

Riverbed Technology
199 Fremont Street
San Francisco, CA 94105

Phone: 415.247.8800
Fax: 415.247.8801
Web: http://www.riverbed.com

Contents

CHAPTER 1 Introduction

Introducing Stingray

The Stingray product family provides high-availability, application-centric traffic management and load balancing solutions. They provide control, intelligence, security and resilience for all your application traffic.

Stingray products are intended for organizations hosting valuable business-critical services, such as TCP and UDP-based services like HTTP (web) and media delivery, and XML-based services such as web services.

Introducing REST

REST (REpresentational State Transfer) is a framework for API design. It is based on generic facilities of the standard HTTP protocol, including the six basic HTTP methods (GET, POST, PUT, DELETE, HEAD, INFO) and the full range of HTTP return codes.

A REST interface partitions the API into a series of 'resources', each of which can be accessed using one or more HTTP methods. (In Stingray, only the GET, PUT, and DELETE methods are used; HEAD, POST and INFO are not currently implemented). Each method operates in Stingray as follows:

- GET: Obtain a representation of the resource, without modifying server state (except perhaps for logging purposes).
- PUT: Create a new resource or apply some change to a resource. Where the resource exists, only those properties specified in the request are modified; all others remain unchanged. If a resource object does not exist, a new one is created.
- DELETE: Delete an existing resource.

Importantly, each resource is uniquely identified with an address, or URI (Uniform Resource Identifier). In other words, if you know the URI you can access the resource (subject to the normal authorization/authentication processes associated with accessing the administrative systems of the traffic manager).

Since all resources have URIs, resources can point to other resources by embedding the URIs of related resources within their representations.

In Stingray, all resources are represented and stored as JSON (JavaScript Object Notation) structures. Requests and responses that interact with the traffic manager through the REST API must adopt the same format.

The full range of HTTP return codes is available in REST, although in practise a useful subset can be identified and applied consistently. So, for example, it should be evident from the response itself whether a request has succeeded or not, without any need for parsing the body of the response. However, Stingray will always attempt to provide extra information regarding a failure into the response body. Please refer to the Errors section of CHAPTER 2 for more details.

Why use a REST API

REST interfaces have become popular in public APIs because of their inherent simplicity. An API can focus on available resources, with details regarding updating and deleting of each resource delegated to the appropriate HTTP method in predictable ways.

The purpose of implementing a REST API is not primarily to add functionality but to add structure. Because of the inherent similarity of all REST APIs (by virtue of their underlying HTTP structure), familiarity with any REST API brings familiarity with all of them. In many cases it is just as easy to implement to a REST design as it is to use a more ad hoc API design, while reaping the benefits that come with well-understood REST conventions.

Finally, the availability of return codes is another example of leveraging known semantics when building a useful API. Without a meaningful return code it becomes necessary to parse every response to find out whether it worked or not. In addition, most modern browsers and Web programming frameworks expect that specific HTTP error codes will be set in the event of error and will respond differently depending on the code. This is especially apparent in the case of AJAX requests, which are often handled differently by many modern Javascript frameworks depending on the status code returned from the server.

A REST-based Architecture

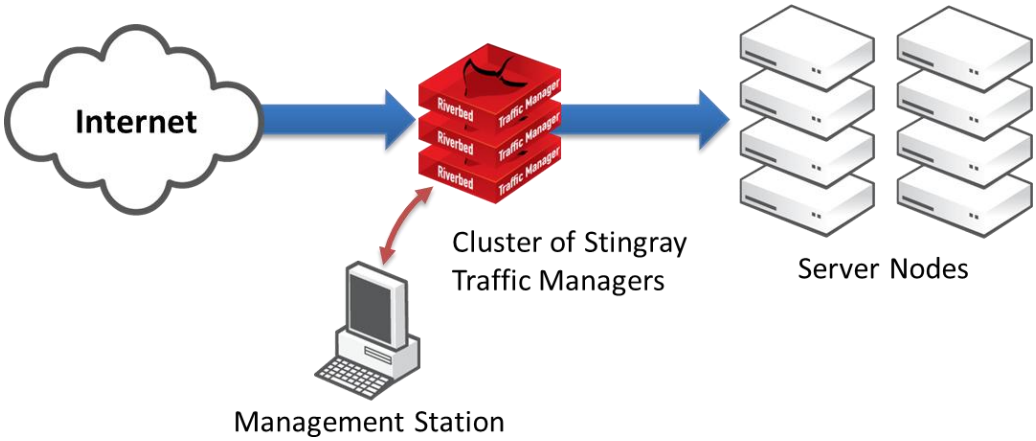

Fig. 1. Arrangement of Management Server, Stingray Cluster and Server Nodes

A cluster of traffic mangers is normally managed using the web-based Administration UI on one of the machines. Stingray's REST API provides an alternative means to remotely administer and configure a Stingray cluster.

The Stingray REST API can be used by any HTTP client or application environment that supports HTTP services.

The REST API is an interface used to configure and manage a cluster of traffic managers remotely.

A management application can issue a REST request to one of the traffic managers in a Stingray cluster. The application may be running on a stand-alone management server, one of the server nodes, or even on one of the traffic managers.

The application can issue the request to any of the Stingray traffic managers. The traffic managers automatically synchronize their configuration, so a configuration change sent to one machine is automatically replicated across the cluster.

Important: Due to the nature of the REST API's ability to access and modify your traffic manager configuration, it is strongly recommended that you disallow access to this service from outside of your trusted network.

Scope of this release

This document describes the features and capabilities of the REST API for the Stingray Traffic Manager 9.2 release. The REST API version referred to in this document is 1.0.

Basic type-checking is performed by the API, however you should ensure that your client application provides suitable validation to ensure the suitability of the configuration data being provided to the traffic manager.

All defined users in the system will have the ability to authenticate a connection through the traffic manager REST API. However, please note that you cannot modify the users configuration file in any way, so it is not currently possible to add/edit/delete users through the API.

A full list of specific features and capabilities supported by this release can be found in the release notes provided with your version.

CHAPTER 2 Typical usage in Stingray

The resource model

The Stingray configuration system is made up of a hierarchy of resources that are manipulated using standard HTTP calls to a listener service running along side the traffic manager. HTTP URIs are used to address the resources in the system.

Each concept in Stingray, such as pools, virtual servers, TrafficScript rules or Service Level Monitoring classes, has an associated resource model. These are represented as JSON structures (MIME type `application/json`), and objects of each resource type are captured in this format.

Typically, a resource follows this format:

```
{
   "properties": {
      "sectionname": {
         "key1": "stringvalue1",
         "key2": numericvalue2,
         "key3": booleanvalue3
      }
   }
}
```

A single instance of a resource, for example a virtual server, will contain a primary group entitled "**properties**". This contains all configuration keys attributable to this resource type.

Sections

The properties group contains a number of sections, one for each *logical set* of keys. There should always be a section entitled "**basic**" containing common configuration items, followed by one or more additional sections according to the specification of the resource.

Sections are designed to contain further configuration keys that have a commonality of purpose or perhaps apply in certain circumstances. For example, monitor classes may have keys that apply only to monitors of particular types.

Data Types

Each `key:value` pair is then presented as a comma-separated list within each section, according to the specification shown throughout this guide. Key names are always delimited by quotes, with the values according to the following rules:

Boolean	A value of `true` or `false` (case-sensitive). For example:
	`"key1": true,`
	`"key2": false`

Int	A numeric positive or negative value with no decimal point. For example:
	`"key1": 1024,`
	`"key2": -10`

Unsigned Int	A numeric positive value with no decimal point. For example:
	`"key1": 0,`
	`"key2": 50`

Float	A numeric positive or negative value that can have a decimal point. For example:
	`"key1": 1.0,`
	`"key2": -1024.111`

String	A set of alpha-numeric characters that may not include new-lines. Non-alpha characters must use correct character escapes. For example:
	`"key1": "Hello world",`
	`"key2": "",`
	`"key3": "Hello y\'all"`

Freeform String	A set of alpha-numeric characters that can contain new-lines. Non-alpha characters must use correct character escapes, and a newline must be represented by a `\n`. For example: `"key1": "Multi-line\nString",`

Password	A string that cannot be read, only written to. When read, it is displayed as a structure that indicates if the password has been set (is non-empty). For example, when reading the key:
	`"key1": { "password_set": false },`
	`"key2": { "password_set": true }`
	When writing to the key, the structure can be unchanged, or a new password can be set:
	`"key2": { "password_set": true },`
	`"key1": "secret123"`

Time	Times are represented as strings in ISO8601 time format, including a time zone designator. For example: `{Year}-{Month}-{Day}T{Hour}:{Minute}:{Second}{Time Zone}`
Set	This is a collection of unique un-ordered items of a particular type, stored as an array. For example: `"key": [ "Item A", "Item B", "Item D" ]`

List	This is a collection of ordered items of a particular type. It may contain duplicates and is stored as a standard array. For example: `"key": [ "Item A", "Item C", "Item A" ]`

Tables	This is a special type designed to allow nested data within a single config key. In some circumstances, you might wish to specify a list/array of data items, such as a list of pool nodes, where each item has one or more extra pieces of configuration data to be attached to it.
	Each one of these nested list entries expects a value known as the **primary key**, used to identify it. Each sub-key value should then be specified in the same way. For example:
	`"key": [`
	`  {`
	`  "prmkey": "Hello World",`
	`  "subkey1": false,`
	`  "subkey2": [ "Item 1", "Item 2" ]`
	`  },`
	`  {`
	`  "prmkey": "Other text",`
	`  "subkey1": true,`
	`  "subkey2": []`
	`  },`
	`]`

Resource URI patterns

All Stingray resources are provided through a common base URI that identifies the root of the resource model. This is:

```
https://<host>:<port>/api/tm/<version>/config/active
```

All traffic manager configuration resources can be found at this point. Instances of a resource type, such as a virtual server, are persistently stored and will alter the host traffic manager's behaviour if changed. Additionally, changes made here are synchronized automatically to all other machines in the cluster.

Note: In the example above, a scheme of HTTPS is used to signify an encrypted connection from a remote client. HTTP is only supported where the connection is to a server on the same host. Please refer to the Authentication section below for more details.

If you wish to view or modify a stored record of a particular resource type, you would append the full path on to the end of this base URI. For example, a request for a virtual server named "Intranet" would look like this:

```
https://myhost:9070/api/tm/1.0/config/active/vservers/Intranet
```

Traversing the tree

Resource URIs can be either:

- configuration resources, or
- a directory structure containing child elements denoting sub-directories or resource *nodes*.

You can test the overall availability of the REST API by querying the following URI:

```
https://<host>:<port>
```

(where `<host>` is the hostname of the traffic manager concerned, and `<port>` is the port that the REST API is published on. For example: http://localhost:9070).

A simple GET request for this URI should yield the following result:

```
{
   "children": [{
      "name": "api",
      "href": "/api/"
   }]
},
```

This shows that `localhost:9070` contains a single child element `"/api"`. We know from the Resource URI patterns section above that the full root URI of the resource model is the following:

```
https://myhost:9070/api/tm/1.0/config/active
```

Therefore, requesting this URI should result in the following list of child elements:

```
{
   "children": [{
      "name": "auth",
      "href": "/api/tm/1.0/config/active/auth"
   }, {
      "name": "actions",
      "href": "/api/tm/1.0/config/active/actions"
   }, {
      "name": "cluster",
      "href": "/api/tm/1.0/config/active/cluster"
   }, {
      "name": "config",
      "href": "/api/tm/1.0/config/active/config"
   }, {
      "name": "vservers",
```

```
      "href": "/api/tm/1.0/config/active/vservers"
   }, {
   ...
   (truncated)
   ...
   }, {
      "name": "zxtms",
      "href": "/api/tm/1.0/config/active/zxtms"
   }]
}
```

This output identifies all resource types available through the traffic manager being queried. Each is identified by a `name` and `href` attribute combination.

A query for a specific resource type will show all instances of that resource defined within the traffic manager configuration. For example, the following URI will list all virtual servers:

```
https://myhost:9070/api/tm/1.0/config/active/vservers
```

The output will show each stored virtual server, as per the following example:

```
{
   "children": [{
      "name": "vs1",
      "href": "/api/tm/1.0/config/active/vservers/vs1"
   }, {
      "name": "vs2",
      "href": "/api/tm/1.0/config/active/vservers/vs2"
   }]
}
```

The REST service

The REST API is a HTTP service running on the traffic manager server. By default it is available on TCP port **9070**, though this can be reconfigured. The REST service supports HTTP versions: 0.9, 1.0, 1.1; Version 1.1 is recommended.

When connecting to the local machine using a loop-back interface (i.e. 127.0.0.1 or localhost), plain HTTP must be used. When connecting from a remote machine, connections must be encrypted using SSL (HTTPS).

The service uses the same SSL certificate as the traffic manager's admin server, which by default is an automatically generated self-signed certificate. Any HTTP client used to connect to the REST API should have the server's self-signed certificate added to its trusted certificate catalogue. Alternatively the admin server/REST certificate can be replaced with one signed by a trusted certificate authority.

Authentication

A REST-based management application communicates with a configuration service running on the Stingray *Admin Server* (the traffic manager-based service used to provide the Admin UI), so the same security considerations apply:

- REST requests are authenticated using HTTP Basic Auth.
- REST traffic over HTTPS is automatically encrypted using SSL. Traffic over HTTP is not encrypted, so should only be used inside a secure environment or to/from *localhost*.
- The Stingray Admin Server will authenticate itself with its SSL certificate, which is generally self-signed. You may need to ensure that your REST application accepts self-signed certificates, or install a trusted SSL certificate in your Stingray admin server.
- REST requests are authenticated using the same user credentials as defined in the Administration Server. Individual object access is synonymous with page access in the Admin UI. For example, if a user wishes to view and manipulate pool objects, they must have been granted access to pools on the access permissions page.

Supported HTTP methods

The REST service supports three primary HTTP methods for accessing and modifying data in the Stingray configuration system:

- GET
- PUT
- DELETE

GET is used when making read-only requests for a resource, whereas PUT is used when updating existing data or adding new configuration objects. DELETE is used when you wish to completely remove configuration objects from the traffic manager. Each of these is covered in more detail below.

Making requests for a resource

A client interacts with the Stingray REST API by performing operations on its resources. An operation is distinguished by the HTTP method used and the path and query components of the URI. Some operations, however, are not applicable to every resource.

The **GET** method is used to retrieve the current representation of the resource it is used on. It does not alter the resource in any way or have any other side effects.

This is achieved by sending a HTTP GET request to the server with no body. The request must accept a response in JSON format only (by specifying an **Accept** header type of `application/json`), and authorization is provided using **HTTP Basic Auth** (see the Authentication section above for more details). Such a request will resemble the following:

```
GET /api/tm/1.0/config/active/bandwidth/BWClass1 HTTP/1.1
Authorization: Basic YWRtaW46c2VjcmV0MTIz
Accept: application/json
```

If successful, the server will return a '200 OK' response code with the full resource in the response body. The above *Bandwidth class* example might produce the following output:

```
{
   "properties": {
      "basic": {
         "maximum": 10000,
         "note": "This is my bandwidth class",
         "sharing": "cluster"
      }
   }
}
```

This is a JSON structure representing the configuration keys present in the requested bandwidth class object. In this case, it consists of a single "basic" section containing three key:value pairs. Other resource types might contain additional sections and corresponding keys.

Setting configuration for a resource

Changing data items in the traffic manager configuration system is achieved through a PUT request. This applies to either **creating** new resource items or **updating** the properties of an existing resource item.

When creating a new resource item, the request URI must contain the full path to the intended item, with the name being the final element of the path. For example, creating a new bandwidth class called "mynewclass" would entail using the following URI:

```
/api/tm/1.0/config/active/bandwidth/mynewclass
```

For both creation and update operations, the request body must contain a representation of the resource properties in JSON format (with the appropriate body "Content-Type" header set). Partial updates to configuration resources can be performed by only including the properties that need to be altered. Other properties will be left as they are.

The REST service returns a "200 OK" response for a correctly updated configuration set, or "201 Created" for establishing a new config object of a particular resource type. In these cases, the full resource is returned as the response body. The only exception to this rule is when updating a raw file, which will instead return a "204 No Content" empty-body response.

Important: You may wish to exercise some care when creating or updating resources. The changes are permanent and no warning will be given for existing configuration that is overridden. If you attempt to create a new resource where one of the same name already exists, you will simply overwrite the properties of the existing record. It is recommended that you build such validation into your REST client application.

Removing resources

A HTTP DELETE request for the full URI of a configuration item can be sent to the REST server to permanently remove it. On success, a "204 No Content" empty-body response is returned.

Further aspects of the resource model

Enumerated types

Some configuration keys can accept one or more of a pre-defined set of values. This is known as an enumerated key type, and the list of possible values (with long description) is provided in the reference guide later in this document.

Uploading files

Resources that represent real files (such as TrafficScript rules) can also be presented in a raw format, where the data returned is the contents of the file. The MIME type of the request payload should be set to `application/octet-stream`.

Errors

If the REST server is unable to handle a HTTP request, it will return a HTTP response with an appropriate HTTP error code. The response body will be in JSON and will contain a data structure describing the error with a unique identifier (different than the numeric error code) and a description.

The unique identifier is made up of 2 parts:

```
{section}.{error_type}
```

Some errors may provide additional formatted information, specified with an optional '`error_info`' parameter. For example, the REST API uses this parameter to return per-property errors when a value fails validation. The following structure demonstrates the general form of an error:

```
{
  "error": {
     "error_id":   "{error identifier}",
     "error_text": "{error description}",
     "error_info": {error specific data structure, optional}
  }
}
```

A validation error occurs when one or more of the properties within a configuration resource fail a validation check. The `error_info` section then contains a sub-error for each property that failed validation. These sub-errors are like normal errors in that they contain an identifier (`error_id`) and a human readable text description (`error_text`):

```
{
  "error": {
```

```
     "error_id":   "resource.validation_failed",
     "error_text": "Some of the properties in the resource
failed validation.",
     "error_info": {
        "basic": {
           "key1": {
              "error_id": "num.range",
              "error_text": "Value must be in range 1000 -
2000."
           }
        }
     }
  }
}
```

Stingray UI Features

Enabling and disabling the API

The REST service can be enabled or disabled from the *REST API* section of the **System > Security** page of the Stingray Admin UI. This page also provides the ability to set the TCP port that the service listens on. The default port is **9070**, however any unreserved port can be used here provided it does not conflict with other services already running on the traffic manager system. The changes are applied as soon as your click **Update**.

Important: The REST API is currently not available in conjunction with the Stingray Multi-Site Manager (MSM) feature. Attempts to enable the REST service whilst MSM is operational will be denied. Equally, attempting to enable MSM whilst the REST service is running will present an error. The current state of the traffic manager remains unchanged in either of these situations.

Controlling timeout events

The *REST API* section of the **System > Security** page additionally provides a number of settings to control how the traffic manager responds to certain timeout events that occur through use of the REST API. These are:

`rest!auth_timeout`	The timeout period, in seconds, for the REST Authentication cache. Each REST request is supplied with user and password credentials as there is no concept of a 'session' in REST. These credentials must be validated each time, but to save requesting repeated external authentications for the same user (from the same IP address) a cache of recent authentications is kept. This timeout is the maximum time a given user can stay in the cache. A setting of 0 disables the cache, forcing every REST request to be authenticated as it is received. However, this will affect the performance

	of the API. (Default: 120 seconds)
`rest!replulltime`	This is the lull time for configuration replication via REST. It is the time, in seconds, of inactivity via the REST API before configuration replication will start. Increasing this value will delay configuration replication among a cluster of traffic managers. (Default: 5 seconds)
`rest!repabstime`	This is the absolute timeout prior to configuration replication via REST. It is the longest time, in seconds, before configuration replication via REST will start, regardless of activity through the API. (Default: 20 seconds)
`rest!reptimeout`	The configuration replication duration timeout via REST. This is the time, in seconds, allowed for the process of configuration replication to complete. On a system with slow cluster communications or a very large configuration, increasing this value will improve replication reliability. (Default: 10 seconds)

Restricting access to trusted users

In addition to username/password access, the *Restricting Access* section of the **System > Security** page provides the ability to further restrict access to the administrative capabilities of your traffic manager system to a set of trusted IP addresses, CIDR subnets or DNS wildcards. Access to the REST API is also affected by this capability.

Log messages in Stingray

The Event Log

A number of specific API-related messages might be found in the Stingray event log under certain conditions:

- `REST API port changed: https://<URI>`

  Raised when the REST Daemon has been asked to change the port it listens on.

- `REST API started: https://<URI>`

Raised when the REST Daemon starts.

- `REST API is shutting down`

  Raised when the REST Daemon closes down.

- `On IPv6 host but cannot set unspecified ip address to ::`

  Raised when the REST Daemon can't set itself up to listen on the IPv6 wildcard address.

- `Could not open traffic manager PID file for read: <error>`

  Raised when REST Daemon can't identify the traffic manager PID, and so can't signal it to reload its config after a change has been made via the REST API.

- `Could not open traffic manager PID file: <error>`

  Raised when REST Daemon can't identify the traffic manager PID, and so can't signal it to reload its config after a change has been made via the REST API.

- `Failed to write to audit log: <error>`

  Raised when the REST Daemon can't add lines to the audit log.

The Audit Log

The audit log records login attempts, configuration changes, and user logouts. It also records changes made using the Stingray Control API, and via the traffic manager CLI. Configuration changes made through the REST API follow the same behavior.

In addition to the typical configuration messages entered into the audit log, Stingray also provides the ability to track user activity in the REST API. It does this by grouping REST request/response exchanges made in close succession from a given user into a ‘session’.

Stingray logs the first request in a group of one or more requests from a particular user/ip address combination in the audit log as a "session start". Requests received after this initial request are deemed to be part of the same user session. Then, after a specified timeout interval since the most recent request was received from the same user, a "session end" is logged.

CHAPTER 3 Examples and use-cases

Typical usage

The following code samples demonstrate how to interact with the REST API for a variety of purposes. The examples are based on Perl using the `REST::Client` module to handle the connections to the traffic manager REST daemon.

Note: Further information on `REST::Client` can be found at the CPAN website: www.cpan.org

A typical Perl client connection might resemble the following:

```
#!/usr/bin/perl

use REST::Client;
use strict;

# Set up the connection
my $client = REST::Client->new();
$client->setHost( 'https://stingrayhost:9070' );
$client->addHeader( 'Authorization', 'Basic YWRtaW46am9iYmll' );
$client->addHeader( 'Content-Type', 'application/json' );

# Perform a HTTP GET on this URI
$client->GET( '/api/tm/1.0/config/active' );

# Print out the response body
print $client->responseContent();
```

In the above example, a new connection is established to the REST service on the traffic manager '`stingrayhost`' on port 9070.

The setHost() function allows us to set up a definitive hostname/port to which all requests will be made. This is an optional feature, and the full hostname can be supplied when making the actual request if multiple hosts are required.

Two HTTP headers can be added here, one to provide *Basic Auth* authentication and the other to provide a declaration of the Content Type when making PUT requests. In the majority of cases, the content type will be '`application/json`', apart from transactions involving raw files where it is necessary to use '`application/octet-stream`'.

A GET request is sent to the REST service with a target of the resource URI as the supplied argument. Typically, the above script will output a JSON structure showing the traffic manager resource tree at the top level:

```
{
    "children": [{
        "name": "rules",
        "href": "/api/tm/1.0/config/active/rules/"
    }, {
```

```
        "name": "actions",
        "href": "/api/tm/1.0/config/active/actions/"
    },
    ...
    (truncated)
    ...
   {
        "name": "auth",
        "href": "/api/tm/1.0/config/active/auth/"
    }]
}
```

Note: Each of the following examples make use of a further Perl module "`JSON`" in order to encode and decode between the JSON string used by `REST::Client` and a native Perl structure. This is done to simplify the parsing algorithm within the script. Further information regarding the `JSON` module can be found on the CPAN website at: www.cpan.org.

List running virtual servers

In this example, we collect data on stored virtual servers by querying the `vservers` resource and identifying which ones are enabled (i.e. running).

The code structure is as follows:

- Instantiate a new REST Client object;
- Specify the hostname/port of the REST service to which all requests are to be directed;
- Add required HTTP headers for authentication and content type;
- Send a GET request for the `vservers` resource in order to return a list of all Virtual Servers on the system;
- Check the response body, and decode from JSON into a Perl structure. This value will be a hash ref;
- Identify the `children` hash key, and iterate through the array to which it points;
- Each array item contains a hash of `name` and `href` associative values;
- Using the `name` value, perform a new GET request to return the full configuration for this named virtual server resource;
- Again, using the decoded JSON response body, identify the Boolean value of the `enabled` key in the `basic` configuration section. If it is `true`, this virtual server is running, so print it's name to STDOUT.

Important: Please note this script does not contain any error checking in order to best demonstrate the basic functionality. It is strongly recommended you incorporate return value checking and other validation mechanisms as appropriate.

```
#!/usr/bin/perl

use REST::Client;
use JSON;
use strict;

# Set up the connection
my $client = REST::Client->new();
$client->setHost( 'https://stingrayhost:9070' );
$client->addHeader( 'Authorization', 'Basic YWRtaW46am9iYmll' );
$client->addHeader( 'Content-Type', 'application/json' );

# Request a list of all virtual servers
client->GET( '/api/tm/1.0/config/active/vservers' );

# Decode response into a perl structure for easy parsing
my $response = decode_json( $client->responseContent() );

# Obtain a reference to the children array
my $vsArrayRef = $response->{children};

# For each VS, make a request for it's configuration and
# check the Boolean value of the 'enabled' key
foreach my $vs ( @$vsArrayRef ) {
   my $vsName = $vs->{name};
   $client->GET( "/api/tm/1.0/config/active/vservers/$vsName" );
   my $vsConfig = decode_json( $client->responseContent() );
   if( $vsConfig->{properties}->{basic}->{enabled} eq "true" ) {
      # Print the name of this matched VS
      print "$vsn\n";
   }
}
```

The expected output of a script such as this would be:

```
$ ./listVS.pl
Main Website
Intranet
Support Site
```

Adding a node to a pool

Provisioning systems can dynamically deploy applications across servers, perhaps in reaction to increased server load. This example demonstrates an application that modifies the nodes that a pool balances traffic to.

The code structure is as follows:

- Instantiate a new REST Client object;
- Specify the hostname/port of the REST service to which all requests are to be directed;
- Add required HTTP headers for authentication and content type;
- Send a GET request for the pool that the new node will be added to. Check the response body, and decode from JSON into a Perl structure. This value will be a hash ref;
- The new node must be added to the list of existing nodes before writing the data back to the pool resource. Failing to do this will result in the existing array being overwritten with a single entry containing the new node name;
- Re-encode the perl structure into JSON and pass as an argument to the PUT request (using the pool name URI as the target);
- In this example, the script performs a check on the response code to ensure any problems are reported back (where the response code is not 200 OK);
- There is an optional portion of code at the end to iterate through the stored node list to ensure the new node name appears.

```
#!/usr/bin/perl -w

use REST::Client;
use JSON;
use strict;

# Set up the connection
my $client = REST::Client->new();
$client->setHost( 'http://localhost:9070' );
$client->addHeader( 'Authorization', 'Basic YWRtaW46am9iYmll' );
$client->addHeader( 'Content-Type', 'application/json' );

# Our pool and new node details
my $poolName = "WebPool";
my $newNode = "www3.riverbed.com:80";

# Get the config for the pool in question
$client->GET( "/api/tm/1.0/config/active/pools/$poolName" );
my $poolConfig = decode_json( $client->responseContent() );

# Find the existing nodes list (a hashref), and add our new node
my $nodesRef = $poolConfig->{properties}->{basic}->{nodes};
push @$nodesRef, $newNode;
```

```
# Re-encode as a JSON string
my $poolStr = encode_json( $poolConfig );

# Now send a PUT request to the REST service
$client->PUT( "/api/tm/1.0/config/active/pools/$poolName",
              $poolStr );

# Print out the response code if we were NOT successful
if( $client->responseCode() ne '200' ) {
   die "FAILED with HTTP code: " . $client->responseCode();
}

# We're done! Verify that the node has been added
$client->GET( "/api/tm/1.0/config/active/pools/$poolName" );
$poolConfig = decode_json( $client->responseContent() );
print "Stored nodes for pool '$poolName':\n";
foreach my $node ( @{$poolConfig->{properties}->{basic}->{nodes}} )
{
   print "$node\n";
}
```

The expected output of a script such as this would be:

```
$ ./addNode.pl
Stored nodes for pool 'WebPool':
www1.riverbed.com:80
www2.riverbed.com:80
www3.riverbed.com:80
```

CHAPTER 4 Resource model reference

About the resource model reference

This chapter lists all the resource types available through the REST API model. The table in each section lists the configuration keys available within that resource, along with the key type and default value (if one exists). There may be a list of permitted values where the key represents an enumerated type, or a description of the Primary and Sub keys where the key is a Table-type property.

The configuration path to use in your URIs is listed above the resource description. For example, the *config path* for a virtual server is `vservers/*`, so to address a stored virtual server named `foo` you would use:

```
/api/tm/1.0/config/active/vservers/foo
```

Action Program

Config path: `actionprogs/*`

This is a program or script that can be referenced and used by actions of type 'Program'

Key	Description
There are no configuration keys to display for this resource.	

Aptimizer Profile

Config path: `aptimizer/profiles/*`

An Aptimizer profile can be applied to a HTTP virtual server to enable automatic web content optimization.

Key	Description
`mode`	Set the Aptimizer mode to turn acceleration on or off. Value type: `Enum(String)` Default: `active`

	Permitted values:	`active`	On - Aptimizer acceleration is enabled
		`idle`	Off - Acceleration is disabled, but requests for Aptimizer resources are served
		`stealth`	Stealth - Acceleration is controlled by a cookie
`show_info_bar`	Show the Aptimizer information bar on aptimized web pages. Value type: `Boolean` Default: `false`		

Aptimizer Application Scope

Config path: `aptimizer/scopes/*`

Application scopes define criteria that match URLs to specific logical web applications hosted by a virtual server.

Key	Description
`hostnames`	The hostnames to limit acceleration to. Value type: `Set(String)` Default: `<none>`
`root`	The root path of the application defined by this application scope. Value type: `String` Default: `/`

Bandwidth Class

Config path: `bandwidth/*`

A Bandwidth class, which can be assigned to a virtual server or pool in order to limit the number of bytes per second used by inbound or outbound traffic.

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td><code>maximum</code></td><td>The maximum bandwidth to allocate to connections that are associated with this bandwidth class (in kbits/second).

Value type: <code>UInt</code>

Default: <code>10000</code></td></tr>
<tr><td><code>note</code></td><td>A description of this bandwidth class.

Value type: <code>FreeformString</code>

Default: <code><none></code></td></tr>
<tr><td><code>sharing</code></td><td>The scope of the bandwidth class.

Value type: <code>Enum(String)</code>

Default: <code>cluster</code>

Permitted values:
<table>
<tr><td><code>cluster</code></td><td>Bandwidth is shared across all traffic managers</td></tr>
<tr><td><code>connection</code></td><td>Each connection can use the maximum rate</td></tr>
<tr><td><code>machine</code></td><td>Bandwidth is shared per traffic manager</td></tr>
</table>
</td></tr>
</table>

Cloud Credentials

Config path: `cloudcredentials/*`

Cloud credentials used in cloud API calls

Key	Description
`api_server`	The vCenter server hostname or IP address. Value type: `String` Default: `<none>`
`cloud_api_timeout`	The traffic manager creates and destroys nodes via API calls. This setting specifies (in seconds) how long to wait for such calls to complete. Value type: `UInt` Default: `200`
`cred1`	The first part of the credentials for the cloud user. Typically this is some variation on the username concept. Value type: `String` Default: `<none>`
`cred2`	The second part of the credentials for the cloud user. Typically this is some variation on the password concept. Value type: `Password` Default: `<none>`
`cred3`	The third part of the credentials for the cloud user. Typically this is some variation on the authentication token concept. Value type: `Password` Default: `<none>`

script	The script to call for communication with the cloud API Value type: String Default: <none>
update_interval	The traffic manager will periodically check the status of the cloud through an API call. This setting specifies the interval between such updates. Value type: UInt Default: 30

Alerting Action

Config path: actions/*

A response to an event occurring in your traffic manager. An example of an action might be sending an email or writing a line to a log file.

Key	Description
note	A description of the action. Value type: FreeformString Default: <none>
timeout	How long the action can run for before it is stopped automatically (set to 0 to disable timeouts). Value type: UInt Default: 60
type	The action type. Value type: Enum(String)

<table>
<tr><td></td><td>Default: <code><none></code>

Permitted values:
<table>
<tr><td><code>email</code></td><td>E-Mail</td></tr>
<tr><td><code>log</code></td><td>Log to File</td></tr>
<tr><td><code>program</code></td><td>Program</td></tr>
<tr><td><code>soap</code></td><td>SOAP Callback</td></tr>
<tr><td><code>syslog</code></td><td>Log to Syslog</td></tr>
<tr><td><code>trap</code></td><td>SNMP Notify or Trap</td></tr>
</table>
</td></tr>
<tr><td><code>verbose</code></td><td>Enable or disable verbose logging for this action.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>email</code> section:</td></tr>
<tr><td><code>server</code></td><td>The SMTP server to which messages should be sent. This must be a valid IPv4 address or resolvable hostname (with optional port).

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>to</code></td><td>A set of e-mail addresses to which messages will be sent.

Value type: <code>Set(String)</code>

Default: <code><none></code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>log</code> section:</td></tr>
<tr><td><code>file</code></td><td>The full path of the file to log to. The text <code>%zeushome%</code> will be replaced with the location where the software is installed.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>from</code></td><td>The e-mail address from which messages will appear to</td></tr>
</table>

<table>
<tr><td></td><td>originate.

Value type: <code>String</code>

Default: <code>stingraytrafficmanager@%hostname%</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>program</code> section:</td></tr>
<tr><td><code>program</code></td><td>The program to run.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>arguments</code></td><td>A table containing arguments and argument values to be passed to the event handling program.

primary key:
<table><tr><td><code>name (String)</code></td><td>The name of the argument to be passed to the event handling program.</td></tr></table>
sub keys:
<table><tr><td><code>value (String)</code></td><td>The value of the argument to be passed to the event handling program.</td></tr><tr><td><code>description (String)</code></td><td>A description for the argument provided to the program.</td></tr></table></td></tr>
<tr><td colspan="2">Configuration keys for the <code>soap</code> section:</td></tr>
<tr><td><code>additional_data</code></td><td>Additional information to send with the SOAP call.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>password</code></td><td>The password for HTTP basic authentication.

Value type: <code>Password</code>

Default: <code><none></code></td></tr>
<tr><td><code>proxy</code></td><td>The address of the server implementing the SOAP interface (For example, https://example.com).</td></tr>
</table>

	Value type: `String` Default: `<none>`
`username`	Username for HTTP basic authentication. Leave blank if you do not wish to use authentication. Value type: `String` Default: `<none>`
Configuration keys for the `syslog` section:	
`sysloghost`	The host and optional port to send syslog messages to (if empty, messages will be sent to localhost). Value type: `String` Default: `<none>`
Configuration keys for the `trap` section:	
`community`	The community string to use when sending a Trap over SNMPv1 or a Notify over SNMPv2c. Value type: `String` Default: `<none>`
`auth_password`	The authentication password for sending a Notify over SNMPv3. Blank to send unauthenticated traps. Value type: `Password` Default: `<none>`
`hash_algorithm`	The hash algorithm for SNMPv3 authentication. Value type: `Enum(String)` Default: `md5` Permitted values: `md5` MD5

<table>
<tr><td></td><td><table><tr><td>sha1</td><td>SHA-1</td></tr></table></td></tr>
<tr><td>priv_password</td><td>The encryption password to encrypt a Notify message for SNMPv3. Requires that authentication also be configured. Blank to send unencrypted traps.

Value type: Password

Default: <none></td></tr>
<tr><td>username</td><td>The SNMP username to use to send the Notify over SNMPv3.

Value type: String

Default: <none></td></tr>
<tr><td>version</td><td>The SNMP version to use to send the Trap/Notify.

Value type: Enum(String)

Default: snmpv1

Permitted values:<table><tr><td>snmpv1</td><td>SNMPv1</td></tr><tr><td>snmpv2c</td><td>SNMPv2c</td></tr><tr><td>snmpv3</td><td>SNMPv3</td></tr></table></td></tr>
<tr><td>traphost</td><td>The hostname or IPv4 address and optional port number that should receive traps.

Value type: String

Default: <none></td></tr>
</table>

Event Type

Config path: `events/*`

Configuration that ties actions to a set of events that trigger them.

Key	Description
`actions`	The actions triggered by events matching this event type, as a list of action references. Value type: `List(Reference(config-event-action))` Default: `<none>`
`built_in`	If set to `Yes` this indicates that this configuration is built-in (provided as part of the software) and must not be deleted or edited. Value type: `Boolean` Default: `false`
`note`	A description of this event type. Value type: `FreeformString` Default: `<none>`
Configuration keys for the `cloudcredentials` section:	
`event_tags`	Cloud credentials event tags Value type: `List(String)` Default: `<none>`
`objects`	Cloud credentials object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `config` section:	

`event_tags`	Configuration file event tags Value type: `List(String)` Default: `<none>`
Configuration keys for the `faulttolerance` section:	
`event_tags`	Fault tolerance event tags Value type: `List(String)` Default: `<none>`
Configuration keys for the `general` section:	
`event_tags`	General event tags Value type: `List(String)` Default: `<none>`
Configuration keys for the `glb` section:	
`event_tags`	GLB service event tags Value type: `List(String)` Default: `<none>`
`objects`	GLB service object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `java` section:	
`event_tags`	Java event tags Value type: `List(String)` Default: `<none>`

Configuration keys for the `licensekeys` section:	
`event_tags`	License key event tags Value type: `List(String)` Default: `<none>`
`objects`	License key object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `locations` section:	
`event_tags`	Location event tags Value type: `List(String)` Default: `<none>`
`objects`	Location object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `monitors` section:	
`event_tags`	Monitor event tags Value type: `List(String)` Default: `<none>`
`objects`	Monitors object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `pools` section:	

`event_tags`	Pool key event tags Value type: `List(String)` Default: `<none>`
`objects`	Pool object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `protection` section:	
`event_tags`	Service protection class event tags Value type: `List(String)` Default: `<none>`
`objects`	Service protection class object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `rules` section:	
`event_tags`	Rule event tags Value type: `List(String)` Default: `<none>`
`objects`	Rule object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `slm` section:	

`event_tags`	SLM class event tags Value type: `List(String)` Default: `<none>`
`objects`	SLM class object names Value type: `List(String)` Default: `<none>`
Configuration keys for the `ssl` section:	
`event_tags`	SSL event tags Value type: `List(String)` Default: `<none>`
Configuration keys for the `sslhw` section:	
`event_tags`	SSL hardware event tags Value type: `List(String)` Default: `<none>`
Configuration keys for the `trafficscript` section:	
`event_tags`	TrafficScript event tags Value type: `List(String)` Default: `<none>`
Configuration keys for the `vservers` section:	
`event_tags`	Virtual server event tags Value type: `List(String)` Default: `<none>`

objects	Virtual server object names Value type: List(String) Default: <none>
Configuration keys for the zxtms section:	
event_tags	Traffic manager event tags Value type: List(String) Default: <none>
objects	Traffic manager object names Value type: List(String) Default: <none>

Extra File

Config path: `extra/*`

A user-uploaded file. Such files can be used in TrafficScript code using the `resource.get` function.

Key	Description
There are no configuration keys to display for this resource.	

GLB Service

Config path: `services/*`

A global load balancing service is used by a virtual server to modify DNS requests in order load balance data across different GLB locations.

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td><code>algorithm</code></td><td>
Defines the global load balancing algorithm to be used.

Value type: <code>Enum(String)</code>

Default: <code>hybrid</code>

Permitted values:
<table>
<tr><td><code>chained</code></td><td>Sends traffic to one location at a time, until that location fails where the next one in the chain is used.</td></tr>
<tr><td><code>geo</code></td><td>Distributes traffic based solely on the geographic location of each client.</td></tr>
<tr><td><code>hybrid</code></td><td>Distribute traffic based on both the load and geographic location.</td></tr>
<tr><td><code>load</code></td><td>Distributes traffic based on the current load to each location.</td></tr>
<tr><td><code>round_robin</code></td><td>Distributes traffic by assigning each request to a new location in turn. Over a period of time, all locations will receive the same number of requests.</td></tr>
<tr><td><code>weighted_random</code></td><td>Distributes traffic in a random way, but according to a weighted policy defined by individual location weights</td></tr>
</table>
</td></tr>
</table>

`all_monitors_needed`	Are all the monitors required to be working in a location to mark this service as alive? Value type: `Boolean` Default: `true`
`chained_auto_failback`	Enable/Disable automatic failback mode. Value type: `Boolean` Default: `false`
`domains`	The domains shown here should be a list of Fully Qualified Domain Names that you would like to balance globally. Responses from the back end DNS servers for queries that do not match this list will be forwarded to the client unmodified. Note: "*" may be used as a wild card. Value type: `Set(String)` Default: `<none>`
`location_draining`	This is the list of locations for which this service is draining. A location that is draining will never serve any of its local IPs for this domain. This can be used to take a location off-line. Value type: `Set(String)` Default: `<none>`
`enabled`	Enable/Disable our response manipulation of DNS Value type: `Boolean` Default: `false`
`geo_effect`	How much should the locality of visitors affect the choice of location used? This value is a percentage, 0% means that no locality information will be used, and 100% means that locality will always control which location is used. Values between the two extremes will act accordingly.

	Value type: `UInt` Default: `50`
`chained_location_order`	The locations this service operates for and defines the order in which locations fail. Value type: `List(String)` Default: `<none>`
`return_ips_on_fail`	Return all or none of the IPs under complete failure Value type: `Boolean` Default: `true`
`rules`	Response rules to be applied in the context of the service, in order, comma separated Value type: `List(Reference(config-trafficscript))` Default: `<none>`
`ttl`	The TTL that should be used for the domains handled by this config, or `-1` if the original TTL should be left as is. Value type: `Int` Default: `-1`
`dnssec_keys`	A table mapping domains to the private keys that authenticate them primary key: `domain (String)` — A domain authenticated by the associated private keys. sub keys: `ssl_key (Set(String))` — Private keys that authenticate the associated domain.
`location_settings`	Table containing location specific settings. primary `location` — Location to which the

<table>
<tr><td></td><td>key: (String) — associated settings apply.

sub keys:<table>
<tr><td>weight (UInt)</td><td>Weight for this location, for use by the weighted random algorithm.</td></tr>
<tr><td>ips (Set(String))</td><td>The IP addresses that are present in a location. If the Global Load Balancer decides to direct a DNS query to this location, then it will filter out all IPs that are not in this list.</td></tr>
<tr><td>monitors (Set(String))</td><td>The monitors that are present in a location.</td></tr>
</table></td></tr>
<tr><td colspan="2">Configuration keys for the log section:</td></tr>
<tr><td>enabled</td><td>Log connections to this GLB service?

Value type: Boolean

Default: false</td></tr>
<tr><td>filename</td><td>The filename the verbose query information should be logged to. Appliances will ignore this.

Value type: String

Default: %zeushome%/zxtm/log/services/%g.log</td></tr>
<tr><td>format</td><td>The format of the log lines

Value type: String

Default: %s %l %q %g %n %d %a</td></tr>
</table>

License

Config path: `licensekeys/*`

A license key is a encoded text file that controls what functionality is available from each traffic manager in the cluster. Every production traffic manager must have a valid licence key in order to function; a traffic manager without a license will operate in developer mode, allowing developers to trial a wide range of functionality, but placing restrictions on bandwidth.

Key	Description
There are no configuration keys to display for this resource.	

Location

Config path: `locations/*`

These are geographic locations as used by **Global Load Balancing** services. Such a location may not necessarily contain a traffic manager; instead it could refer to the location of a remote datacenter.

Key	Description
`id`	The identifier of this location. Value type: `UInt` Default: `<none>`
`latitude`	The latitude of this location. Value type: `Float` Default: `0.0`
`longitude`	The longitude of this location. Value type: `Float` Default: `0.0`
`note`	A note, used to describe this location.

	Value type: `FreeformString` Default: `<none>`
`type`	Does this location contain traffic managers and configuration or is it a recipient of GLB requests? Value type: `Enum(String)` Default: `config` Permitted values: `config` — Configuration `glb` — GLB

Monitor

Config path: `monitors/*`

Monitors check important remote services are running, by periodically sending them traffic and checking the response is correct. They are used by virtual servers to detect the failure of backend nodes.

Key	Description
`back_off`	Should the monitor slowly increase the delay after it has failed? Value type: `Boolean` Default: `true`
`delay`	The minimum time between calls to a monitor. Value type: `UInt` Default: `3`
`failures`	The number of times in a row that a node must fail execution of the monitor before it is classed as unavailable.

<table>
<tr><td></td><td>Value type: UInt

Default: 3</td></tr>
<tr><td>machine</td><td>The machine to monitor, where relevant this should be in the form <hostname>:<port>, for "ping" monitors the :<port> part must not be specified.

Value type: String

Default: <none></td></tr>
<tr><td>note</td><td>A description of the montitor.

Value type: FreeformString

Default: <none></td></tr>
<tr><td>scope</td><td>A monitor can either monitor each node in the pool separately and disable an individual node if it fails, or it can monitor a specific machine and disable the entire pool if that machine fails. GLB location monitors must monitor a specific machine.

Value type: Enum(String)

Default: pernode

Permitted values:<table><tr><td>pernode</td><td>Node: Monitor each node in the pool separately</td></tr><tr><td>poolwide</td><td>Pool/GLB: Monitor a specified machine</td></tr></table></td></tr>
<tr><td>timeout</td><td>The maximum runtime for an individual instance of the monitor.

Value type: UInt

Default: 3</td></tr>
<tr><td>type</td><td>The internal monitor implementation of this monitor.

Value Enum(String)</td></tr>
</table>

<table>
<tr><td></td><td>type:

Default: <code>ping</code>

Permitted values:
<table>
<tr><td><code>connect</code></td><td>TCP Connect monitor</td></tr>
<tr><td><code>http</code></td><td>HTTP monitor</td></tr>
<tr><td><code>ping</code></td><td>Ping monitor</td></tr>
<tr><td><code>program</code></td><td>External program monitor</td></tr>
<tr><td><code>rtsp</code></td><td>RTSP monitor</td></tr>
<tr><td><code>sip</code></td><td>SIP monitor</td></tr>
<tr><td><code>tcp_transaction</code></td><td>TCP transaction monitor</td></tr>
</table>
</td></tr>
<tr><td><code>use_ssl</code></td><td>Whether or not the monitor should connect using SSL.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>verbose</code></td><td>Whether or not the monitor should emit verbose logging. This is useful for diagnosing problems.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>http</code> section:</td></tr>
<tr><td><code>authentication</code></td><td>The HTTP basic-auth <code><user>:<password></code> to use for the test HTTP request.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>body_regex</code></td><td>A regular expression that the HTTP response body must match. If the response body content doesn't matter then set this to <code>.*</code> (match anything).

Value <code>String</code></td></tr>
</table>

	type: Default: `<none>`
`host_header`	The host header to use in the test HTTP request. Value type: `String` Default: `<none>`
`path`	The path to use in the test HTTP request. This must be a string beginning with a `/` (forward slash). Value type: `String` Default: `/`
`status_regex`	A regular expression that the HTTP status code must match. If the status code doesn't matter then set this to `.*` (match anything). Value type: `String` Default: `^[234][0-9][0-9]$`
Configuration keys for the `rtsp` section:	
`body_regex`	The regular expression that the RTSP response body must match. Value type: `String` Default: `<none>`
`path`	The path to use in the RTSP request (some servers will return 500 Internal Server Error unless this is a valid media file). Value type: `String` Default: `/`
`status_regex`	The regular expression that the RTSP response status code must match.

<table>
<tr><td></td><td>Value type: <code>String</code>

Default: <code>^[234][0-9][0-9]$</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>script</code> section:</td></tr>
<tr><td><code>program</code></td><td>The program to run. This must be an executable file, either within the monitor scripts directory or specified as an absolute path to some other location on the filesystem.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>arguments</code></td><td>A table containing arguments and argument values to be passed to the monitor program.

primary key:<table><tr><td><code>name (String)</code></td><td>The name of the argument to be passed to the monitor program.</td></tr></table>
sub keys:<table><tr><td><code>value (String)</code></td><td>The value of the argument to be passed to the monitor program.</td></tr><tr><td><code>description (String)</code></td><td>A description for the argument provided to the program.</td></tr></table></td></tr>
<tr><td colspan="2">Configuration keys for the <code>sip</code> section:</td></tr>
<tr><td><code>body_regex</code></td><td>The regular expression that the SIP response body must match.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>status_regex</code></td><td>The regular expression that the SIP response status code must match.

Value type: <code>String</code>

Default: <code>^[234][0-9][0-9]$</code></td></tr>
</table>

<table>
<tr><td><code>transport</code></td><td>Which transport protocol the SIP monitor will use to query the server.

Value type: <code>Enum(String)</code>

Default: <code>udp</code>

Permitted values:<table><tr><td><code>tcp</code></td><td>TCP</td></tr><tr><td><code>udp</code></td><td>UDP</td></tr></table></td></tr>
<tr><td colspan="2">Configuration keys for the <code>tcp</code> section:</td></tr>
<tr><td><code>close_string</code></td><td>An optional string to write to the server before closing the connection.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>max_response_len</code></td><td>The maximum amount of data to read back from a server, use 0 for unlimited. Applies to TCP and HTTP monitors.

Value type: <code>UInt</code>

Default: <code>2048</code></td></tr>
<tr><td><code>response_regex</code></td><td>A regular expression to match against the response from the server. Applies to TCP monitors only.

Value type: <code>String</code>

Default: <code>.+</code></td></tr>
<tr><td><code>write_string</code></td><td>The string to write down the TCP connection.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>udp</code> section:</td></tr>
<tr><td><code>accept_all</code></td><td>If this monitor uses UDP, should it accept responses from any IP and port?</td></tr>
</table>

	Value type: `Boolean` Default: `false`

Monitor Program

Config path: `scripts/*`

An executable program that can be used to by external program monitors to report the health of backend services.

Key	Description
There are no configuration keys to display for this resource.	

Session Persistence Class

Config path: `persistence/*`

A session persistence class is used to identify the session a new connection belongs too and deliver it to the same backend node.

Key	Description
`cookie`	The cookie name to use for tracking session persistence. Value type: `String` Default: `<none>`
`delete`	Whether or not the session should be deleted when a session failure occurs. (Note, setting a failure mode of 'choose a new node' implicitly deletes the session.) Value type: `Boolean` Default: `true`

<table>
<tr><td>failure_mode</td><td>The action the pool should take if the session data is invalid or it cannot contact the node specified by the session.

Value type: Enum(String)

Default: new_node

Permitted values:<table><tr><td>close</td><td>Close the connection (using the Virtual Servers error file)</td></tr><tr><td>new_node</td><td>Choose a new node to use</td></tr><tr><td>url</td><td>Redirect the user to a given URL</td></tr></table></td></tr>
<tr><td>note</td><td>A description of the session persistence class.

Value type: FreeformString

Default: <none></td></tr>
<tr><td>type</td><td>The type of session persistence to use.

Value type: Enum(String)

Default: ip

Permitted values:<table><tr><td>asp</td><td>ASP and ASP.NET session persistence</td></tr><tr><td>cookie</td><td>Monitor application cookies</td></tr><tr><td>ip</td><td>IP-based persistence</td></tr><tr><td>j2ee</td><td>J2EE session persistence</td></tr><tr><td>named</td><td>Named Node session persistence</td></tr><tr><td>ssl</td><td>SSL Session ID persistence</td></tr><tr><td>transparent</td><td>Transparent session affinity</td></tr><tr><td>universal</td><td>Universal session persistence</td></tr><tr><td>x_zeus</td><td>X-Zeus-Backend cookies</td></tr></table></td></tr>
<tr><td>url</td><td>The redirect URL to send clients to if the session persistence</td></tr>
</table>

	is configured to redirect users when a node dies. Value type: `String` Default: `<none>`

Pool

Config path: `pools/*`

A pool manages a group of backend nodes. It routes traffic to the most appropriate node, based on load balancing and session persistence criteria.

Key	Description
`bandwidth_class`	The Bandwidth Management Class this pool uses, if any. Value type: `Reference(config-bandwidth)` Default: `<none>`
`disabled`	A list of nodes in the pool that are in the 'disabled' state. Value type: `Set(String)` Default: `<none>`
`draining`	A list of nodes in the pool that are in the 'draining' state. Value type: `Set(String)` Default: `<none>`
`failure_pool`	If all of the nodes in this pool have failed, then requests can be diverted to another pool. Value type: `Reference(config-pool)` Default: `<none>`

max_idle_connections_pernode	The maximum number of unused HTTP keepalive connections that should be maintained to an individual node. Zero signifies no limit. Value type: UInt Default: 50
monitors	The monitors assigned to this pool, used to detect failures in the back end nodes. Value type: Set(Reference(config-monitor)) Default: <none>
node_connection_attempts	The number of times the software will attempt to connect to the same back-end node before marking it as failed. This is only used when passive_monitoring is enabled. Value type: UInt Default: 3
nodes	A list of all active and draining nodes in this pool. A node should be specified as a <ip>:<port> pair. Value type: Set(String) Default: <none>
note	A description of the pool. Value type: String Default: <none>
passive_monitoring	Whether or not the software should check that 'real' requests (i.e. not those from monitors) to this pool appear to be working. This should normally be enabled, so that when a node is refusing connections, responding too slowly, or sending back invalid data, it can mark that node as failed, and stop sending requests to it. If this is disabled, you should ensure that suitable health monitors are configured to check your servers instead,

	otherwise failed requests will not be detected and subsequently retried. Value type: `Boolean` Default: `true`
`persistence_class`	The default Session Persistence class this pool uses, if any. Value type: `Reference(config-persistence)` Default: `<none>`
`transparent`	Whether or not connections to the back-ends appear to originate from the source client IP address. Value type: `Boolean` Default: `false`
Configuration keys for the `auto_scaling` section:	
`cloud_credentials`	The Cloud Credentials object containing authentication credentials to use in cloud API calls. Value type: `Reference(cloud-api)` Default: `<none>`
`cluster`	The ESX host or ESX cluster name to put the new virtual machine instances on. Value type: `String` Default: `<none>`
`data_center`	The name of the logical datacenter on the vCenter server. Virtual machines will be scaled up and down under the datacenter root folder. Value type: `String` Default: `<none>`

`data_store`	The name of the datastore to be used by the newly created virtual machine. Value type: `String` Default: `<none>`
`enabled`	Are the nodes of this pool subject to auto-scaling? If yes, nodes will be automatically added and removed from the pool by the chosen auto-scaling mechanism. Value type: `Boolean` Default: `false`
`external`	Whether or not auto-scaling is being handled by an external system. Set this value to true if all aspects of auto-scaling are handled by an external system, such as RightScale. If set to false, the traffic manager will determine when to scale the pool and will communicate with the cloud provider to create and destroy nodes as necessary. Value type: `Boolean` Default: `true`
`hysteresis`	The time period in seconds for which a change condition must persist before the change is actually instigated. Value type: `UInt` Default: `20`
`imageid`	The identifier for the image of the instances to create. Value type: `String` Default: `<none>`
`ips_to_use`	Which type of IP addresses on the node to use. Choose private IPs if the traffic manager is in the same cloud as the nodes, otherwise choose public IPs. Value `Enum(String)`

<table>
<tr><td></td><td>type:

Default: <code>publicips</code>

Permitted values:
<table>
<tr><td><code>private_ips</code></td><td>Private IP addresses</td></tr>
<tr><td><code>publicips</code></td><td>Public IP addresses</td></tr>
</table>
</td></tr>
<tr><td><code>last_node_idle_time</code></td><td>The time in seconds for which the last node in an auto-scaled pool must have been idle before it is destroyed. This is only relevant if min_nodes is 0.

Value type: <code>UInt</code>

Default: <code>3600</code></td></tr>
<tr><td><code>max_nodes</code></td><td>The maximum number of nodes in this auto-scaled pool.

Value type: <code>UInt</code>

Default: <code>4</code></td></tr>
<tr><td><code>min_nodes</code></td><td>The minimum number of nodes in this auto-scaled pool.

Value type: <code>UInt</code>

Default: <code>1</code></td></tr>
<tr><td><code>name</code></td><td>The beginning of the name of nodes in the cloud that are part of this auto-scaled pool

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>port</code></td><td>The port number to use for each node in this auto-scaled pool.

Value type: <code>UInt</code>

Default: <code>80</code></td></tr>
<tr><td><code>refractory</code></td><td>The time period in seconds after the instigation of a re-size during which no further changes will be made to the pool</td></tr>
</table>

	size. Value type: `UInt` Default: `180`
`response_time`	The expected response time of the nodes in ms. This time is used as a reference when deciding whether a node's response time is conforming. All responses from all the nodes will be compared to this reference and the percentage of conforming responses is the base for decisions about scaling the pool up or down. Value type: `UInt` Default: `1000`
`scale_down_level`	The fraction, in percent, of conforming requests above which the pool size is decreased. If the percentage of conforming requests exceeds this value, the pool is scaled down. Value type: `UInt` Default: `95`
`scale_up_level`	The fraction, in percent, of conforming requests below which the pool size is increased. If the percentage of conforming requests drops below this value, the pool is scaled up. Value type: `UInt` Default: `40`
`size_id`	The identifier for the size of the instances to create. Value type: `String` Default: `<none>`
Configuration keys for the `connection` section:	
`max_connect_time`	How long the pool should wait for a connection to a node to be established before giving up and trying another node.

	Value type: `UInt` Default: `4`
`max_connections_per_node`	The maximum number of concurrent connections allowed to each back-end node in this pool per machine. A value of 0 means unlimited connections. Value type: `UInt` Default: `<none>`
`max_queue_size`	The maximum number of connections that can be queued due to connections limits. A value of 0 means unlimited queue size. Value type: `UInt` Default: `<none>`
`max_reply_time`	How long the pool should wait for a response from the node before either discarding the request or trying another node (retryable requests only). Value type: `UInt` Default: `30`
`queue_timeout`	The maximum time to keep a connection queued in seconds. Value type: `UInt` Default: `10`
Configuration keys for the `ftp` section:	
`support_rfc_2428`	Whether or not the backend IPv4 nodes understand the EPRT and EPSV command from RFC 2428. It is always assumed that IPv6 nodes support these commands. Value type: `Boolean` Default: `false`

<table>
<tr><td colspan="3">Configuration keys for the <code>http</code> section:</td></tr>
<tr><td><code>keepalive</code></td><td colspan="2">Whether or not the pool should maintain HTTP keepalive connections to the nodes.

Value type: <code>Boolean</code>

Default: <code>true</code></td></tr>
<tr><td><code>keepalive_non_idempotent</code></td><td colspan="2">Whether or not the pool should maintain HTTP keepalive connections to the nodes for non-idempotent requests.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td colspan="3">Configuration keys for the <code>load_balancing</code> section:</td></tr>
<tr><td><code>algorithm</code></td><td colspan="2">The load balancing algorithm that this pool uses to distribute load across its nodes.

Value type: <code>Enum(String)</code>

Default: <code>round_robin</code>

Permitted values:</td></tr>
<tr><td></td><td><code>fastest_response_time</code></td><td>The Response Time algorithm monitors the response times for recent requests to each node. It sends each new request to the node that has recently been responding the most quickly.</td></tr>
</table>

		`least_connections`	This algorithm sends each new request to the node with the fewest currently active connections.
		`perceptive`	The Perceptive algorithm uses a combination of response time data and connection counts to predict which node is likely to have the fastest response time for each request.
		`random`	This algorithm chooses a random node for each request.
		`round_robin`	This algorithm distributes traffic by assigning each request to a new node in turn.

		`weighted_least_connections`	This algorithm works in a similar way to the Least Connections algorithm, but assigns more requests to nodes with a greater 'weight'.
		`weighted_round_robin`	Weighted Round Robin works in a similar way to Round Robin, but assigns more requests to nodes with a greater 'weight'.
`priority_enabled`	Enable priority lists. Value type: `Boolean` Default: `false`		
`priority_nodes`	Minimum number of highest-priority active nodes. Value type: `UInt` Default: `1`		
`priority_values`	A list of node priorities, higher values signify higher priority. Priorities are specified using the format `<ip>:<port>:<priority>`, if a priority is not specified for a node it is assumed to be `1`. Value type: `Set(String)`		

<table>
<tr><td></td><td>Default: <code><none></code></td></tr>
<tr><td><code>node_weighting</code></td><td>A table containing per-node weighting for use in some load balancing algorithms (weighted least connections and weighted round robin).
primary key: <code>node (String)</code> — Node to which the weighting should be applied.
sub keys: <code>weight (Int)</code> — Weight for the node. The actual value in isolation does not matter: As long as it is a valid integer 1-100, the per-node weightings are calculated on the relative values between the nodes.</td></tr>
<tr><td colspan="2">Configuration keys for the <code>node</code> section:</td></tr>
<tr><td><code>close_on_death</code></td><td>Close all connections to a node once we detect that it has failed.
Value type: <code>Boolean</code>
Default: <code>false</code></td></tr>
<tr><td><code>retry_fail_time</code></td><td>The amount of time, in seconds, that a traffic manager will wait before re-trying a node that has been marked as failed by passive monitoring.
Value type: <code>UInt</code>
Default: <code>60</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>smtp</code> section:</td></tr>
<tr><td><code>send_starttls</code></td><td>If we are encrypting traffic for an SMTP connection, should we upgrade to SSL using STARTTLS.
Value type: <code>Boolean</code>
Default: <code>true</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>ssl</code> section:</td></tr>
</table>

`client_auth`	Whether or not a suitable certificate and private key from the SSL Client Certificates catalog be used if the back-end server requests client authentication. Value type: `Boolean` Default: `false`
`enable`	Whether or not the pool should encrypt data before sending it to a back-end node. Value type: `Boolean` Default: `false`
`enhance`	SSL protocol enhancements allow your traffic manager to prefix each new SSL connection with information about the client. This enables Riverbed Web Servers to run multiple SSL sites, and to discover the client's IP address. Only enable this if you are using nodes for this pool which are Riverbed Web Servers or Stingray Traffic Managers, whose virtual servers have the `ssl_trust_magic` setting enabled. Value type: `Boolean` Default: `false`
`send_close_alerts`	Whether or not to send an SSL/TLS "close alert" when initiating a socket disconnection. Value type: `Boolean` Default: `false`
`server_name`	Whether or not the software should use the TLS 1.0 server_name extension, which may help the back-end node provide the correct certificate. Enabling this setting will force the use of at least TLS 1.0. Value type: `Boolean` Default: `false`

<table>
<tr><td><code>strict_verify</code></td><td>Whether or not strict certificate verification should be performed. This will turn on checks to disallow server certificates that don't match the server name, are self-signed, expired, revoked, or have an unknown CA.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>tcp</code> section:</td></tr>
<tr><td><code>nagle</code></td><td>Whether or not Nagle's algorithm should be used for TCP connections to the back-end nodes.

Value type: <code>Boolean</code>

Default: <code>true</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>udp</code> section:</td></tr>
<tr><td><code>accept_from</code></td><td>The IP addresses and ports from which responses to UDP requests should be accepted.
If set to accept responses from a specific set of IP addresses, you will need to enter a CIDR Mask (such as 10.100.0.0/16).

Value type: <code>Enum(String)</code>

Default: <code>dest_only</code>

Permitted values:<table><tr><td><code>all</code></td><td>Any IP address and any port</td></tr><tr><td><code>dest_ip_only</code></td><td>Only the IP address to which the request was sent, but from any port</td></tr><tr><td><code>dest_only</code></td><td>Only the IP address and port to which the request was sent</td></tr><tr><td><code>ip_mask</code></td><td>Only a specific set of IP addresses, but from any port</td></tr></table></td></tr>
<tr><td><code>accept_from_mask</code></td><td>The CIDR mask that matches IPs we want to receive responses from.</td></tr>
</table>

	Value type: `String` Default: `<none>`

Protection Class

Config path: `protection/*`

A protection class specifies the level of protection against network attacks for a virtual server.

Key	Description
`debug`	Whether or not to output verbose logging. Value type: `Boolean` Default: `false`
`enabled`	Enable or disable this service protection class. Value type: `Boolean` Default: `true`
`log_time`	Log service protection messages at these intervals. If set to `0` no messages will be logged and no alerts will be sent. Value type: `UInt` Default: `60`
`note`	A description of the service protection class. Value type: `String` Default: `<none>`
`rule`	A TrafficScript rule that will be run on the connection after the service protection criteria have been evaluated. This rule will be executed prior to normal rules configured for the

	virtual server. Value type: `Reference(config-trafficscript)` Default: `<none>`
`testing`	Place the service protection class into testing mode. (Log when this class would have dropped a connection, but allow all connections through.) Value type: `Boolean` Default: `false`
Configuration keys for the `access_restriction` section:	
`allowed`	Always allow access to these IP addresses. This overrides the connection limits for these machines, but does not stop other restrictions such as HTTP validity checks. Value type: `Set(String)` Default: `<none>`
`banned`	Disallow access to these IP addresses. Value type: `Set(String)` Default: `<none>`
Configuration keys for the `connection_limiting` section:	
`max_10_connections`	Maximum simultaneous connections allowed from the top ten busiest IP addresses. Value type: `UInt` Default: `200`
`max_1_connections`	Maximum simultaneous connections allowed from one IP address. Value type: `UInt`

	Default: `30`
`max_connection_rate`	Maximum number of connections from one IP address in the `rate_timer` interval. Set to `0` to make this unlimited. If applied to an HTTP Virtual Server each request sent on a connection that is kept alive will also be considered. Value type: `UInt` Default: `<none>`
`min_connections`	Always allow at least this number of simultaneous connections from each IP address before applying restrictions. Set to `0` to allow unlimited simultaneous connections. Value type: `UInt` Default: `4`
`rate_timer`	How frequently the `max_connection_rate` is assessed. For example, a value of `1` (second) will impose a limit of `max_connection_rate` connections per *second*; a value of `60` will impose a limit of `max_connection_rate` connections per *minute*. Value type: `UInt` Default: `60`
Configuration keys for the `http` section:	
`check_rfc2396`	Whether or not requests with poorly-formed URLs be should be rejected. This tests URL compliance as defined in RFC2396. Note that enabling this may block some older, non-conforming web browsers. Value type: `Boolean` Default: `false`
`max_body_length`	Maximum permitted length of HTTP request body data, set to `0` to disable the limit. Value `UInt`

	type: Default: `<none>`
`max_header_length`	Maximum permitted length of a single HTTP request header (key and value), set to `0` to disable the limit. Value type: `UInt` Default: `<none>`
`max_request_length`	Maximum permitted size of all the HTTP request headers, set to `0` to disable the limit. Value type: `UInt` Default: `<none>`
`max_url_length`	Maximum permitted URL length, set to `0` to disable the limit. Value type: `UInt` Default: `<none>`
`reject_binary`	Whether or not URLs and HTTP request headers that contain binary data (after decoding) should be rejected. Value type: `Boolean` Default: `false`
`send_error_page`	This setting tells the traffic manager to send an HTTP error message if a connection fails the service protection tests, instead of just dropping it. Details of which HTTP response will be sent when particular tests fail can be found in the Help section for this page. Value type: `Boolean` Default: `true`

Rate Shaping Class

Config path: `rate/*`

A rate shaping class restricts the number of connections being processed by a virtual server at once.

Key	Description
`max_rate_per_minute`	Requests that are associated with this rate class will be rate-shaped to this many requests per minute, set to `0` to disable the limit. Value type: `UInt` Default: `<none>`
`max_rate_per_second`	Although requests will be rate-shaped to the `max_rate_per_minute`, the traffic manager will also rate limit per-second. This smooths traffic so that a full minute's traffic will not be serviced in the first second of the minute, set this to `0` to disable the per-second limit. Value type: `UInt` Default: `<none>`
`note`	A description of the rate class. Value type: `FreeformString` Default: `<none>`

Security Settings

Config path: `security`

Security settings that restrict remote administration for the cluster. Additional security options can be found in Global Settings.

Key	Description

access	Access to the admin server and REST API is restricted by usernames and passwords. You can further restrict access to just trusted IP addresses, CIDR IP subnets or DNS wildcards. These access restrictions are also used when another traffic manager initially joins the cluster, after joining the cluster these restrictions are no longer used. Care must be taken when changing this setting, as it can cause the administration server to become inaccessable.Access to the admin UI will not be affected until it is restarted. Value type: `Set(String)` Default: `<none>`

Global Settings

Config path: `settings.cfg`

General settings that apply to every machine in the cluster.

Key	Description
accepting_delay	How often, in milliseconds, each traffic manager child process (that isn't listening for new connections) checks to see whether it should start listening for new connections. Value type: `UInt` Default: `50`
chunk_size	The default chunk size for reading/writing requests. Value type: `UInt` Default: `4096`
client_first_opt	Whether or not your traffic manager should make use of TCP optimisations to defer the processing of new client-first connections until the client has sent some data. Value type: `Boolean` Default: `false`

max_fds	The maximum number of file descriptors that your traffic manager will allocate. Value type: UInt Default: 1048576
monitor_memory_size	The maximum number of nodes that can be monitored. This is used to size the shared memory, that keeps track of the state. Value type: UInt Default: 4096
rate_class_limit	The maximum number of Rate classes that can be created. Approximately 100 bytes will be pre-allocated per Rate class. Value type: UInt Default: 25000
shared_pool_size	The size of the shared memory pool used for shared storage across worker processes (e.g. bandwidth shared data).This is specified as either a percentage of system RAM, 5% for example, or an absolute size such as 10MB. Value type: String Default: 10MB
slm_class_limit	The maximum number of SLM classes that can be created. Approximately 100 bytes will be pre-allocated per SLM class. Value type: UInt Default: 1024
so_rbuff_size	The size of the operating system's read buffer. A value of 0 (zero) means to use the OS default; in normal circumstances this is what should be used. Value type: UInt Default: <none>
so_wbuff_size	The size of the operating system's write buffer. A value of 0 (zero) means to use the OS default; in normal circumstances this is what should be used.

<table>
<tr><td></td><td>Value type: UInt

Default: <none></td></tr>
<tr><td>socket_optimizations</td><td>Whether or not the traffic manager should use potential network socket optimisations. If set to auto, a decision will be made based on the host platform.

Value type: Enum(String)

Default: auto

Permitted values:<table><tr><td>auto</td><td>Decide based on local platform</td></tr><tr><td>no</td><td>Disable socket optimizations</td></tr><tr><td>yes</td><td>Enable socket optimizations</td></tr></table></td></tr>
<tr><td>storage_shared</td><td>Whether the storage for the traffic managers' configuration is shared between cluster members.

Value type: Boolean

Default: false</td></tr>
<tr><td>tip_class_limit</td><td>The maximum number of Traffic IP Groups that can be created.

Value type: UInt

Default: 10000</td></tr>
<tr><td colspan="2">Configuration keys for the admin section:</td></tr>
<tr><td>ssl_insert_extra_fragment</td><td>Whether or not SSL3 and TLS1 use one-byte fragments as a BEAST countermeasure for admin server and internal connections.

Value type: Boolean

Default: false</td></tr>
<tr><td>ssl3_allow_rehandshake</td><td>Whether or not SSL3/TLS re-handshakes should be supported for admin server and internal connections.

Value type: Enum(String)

Default: rfc5746

Permitted values:<table><tr><td>always</td><td>Always allow</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td>never</td><td>Never allow</td></tr>
<tr><td>rfc5746</td><td>Only if client uses RFC 5746 (Secure Renegotiation Extension)</td></tr>
<tr><td>safe</td><td>Allow safe re-handshakes</td></tr>
</table></td></tr>
<tr><td>ssl3_ciphers</td><td>The SSL ciphers to use for admin server and internal connections. For information on supported ciphers see the online help.

Value type: String

Default: SSL_RSA_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_AES_256_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA</td></tr>
<tr><td>ssl3_diffie_hellman_key_length</td><td>The length in bits of the Diffie-Hellman key for ciphers that use Diffie-Hellman key agreement for admin server and internal connections.

Value type: Enum(UInt)

Default: dh_2048

Permitted values:<table>
<tr><td>dh_1024</td><td>Use 1024 bit keys for Diffie-Hellman ciphers.</td></tr>
<tr><td>dh_2048</td><td>Use 2048 bit keys for Diffie-Hellman ciphers.</td></tr>
<tr><td>dh_3072</td><td>Use 3072 bit keys for Diffie-Hellman ciphers.</td></tr>
<tr><td>dh_4096</td><td>Use 4096 bit keys for Diffie-Hellman ciphers.</td></tr>
</table></td></tr>
<tr><td>ssl_max_handshake_message_size</td><td>The maximum size (in bytes) of SSL handshake messages that the admin server and internal connections will accept. To accept any size of handshake message the key should be set to the value 0.

Value type: UInt

Default: 10240</td></tr>
<tr><td>support_ssl2</td><td>Whether or not SSL2 support is enabled for admin server and internal connections.

Value type: Boolean</td></tr>
</table>

	Default: `false`
`support_ssl3`	Whether or not SSL3 support is enabled for admin server and internal connections. Value type: `Boolean` Default: `false`
`support_tls1`	Whether or not TLS1.0 support is enabled for admin server and internal connections. Value type: `Boolean` Default: `true`
`support_tls11`	Whether or not TLS1.1 support is enabled for admin server and internal connections. Value type: `Boolean` Default: `true`
Configuration keys for the `appliance` section:	
`bootloader_password`	The password used to protect the bootloader. An empty string means there will be no protection. Value type: `Password` Default: `<none>`
`manage_ncipher`	Whether or not we should manage the nCipher Support Software automatically. Value type: `Boolean` Default: `true`
`nethsm_ip`	The IP address of the nCipher NetHSM to use. Value type: `String` Default: `<none>`
`nethsm_esn`	The ESN (electronic serial number) for the NetHSM. Value type: `String`

	Default: <none>
nethsm_hash	The key hash for the NetHSM. Value type: String Default: <none>
nethsm_ncipher_rfs	The IP address of the nCipher Remote File System to use. Value type: String Default: <none>
Configuration keys for the cluster_comms section:	
allow_update_default	The default value of allow_update for new cluster members. If you have cluster members joining from less trusted locations (such as cloud instances) this can be set to false in order to make them effectively "read-only" cluster members. Value type: Boolean Default: true
allowed_update_hosts	The hosts that can contact the internal administration port on each traffic manager. This should be a list containing IP addresses, CIDR IP subnets, and localhost; or it can be set to all to allow any host to connect. Value type: String Default: all
state_sync_interval	How often to propagate the session persistence and bandwidth information to other traffic managers in the same cluster. Set this to 0 (zero) to disable propagation. Note that a cluster using "unicast" heartbeat messages cannot turn off these messages. Value type: UInt Default: 3
state_sync_timeout	The maximum amount of time to wait when propagating session persistence and bandwidth information to other traffic managers in the same cluster. Once this timeout is hit the transfer is aborted and a new connection created. Value type: UInt

	Default: `6`
Configuration keys for the `connection` section:	
`idle_timeout`	How long an unused HTTP keepalive connection should be kept before it is discarded. Value type: `UInt` Default: `10`
`listen_queue_size`	The listen queue size for managing incoming connections. It may be necessary to increase the system's listen queue size if this value is altered. If the value is set to `0` then the default system setting will be used. Value type: `UInt` Default: `<none>`
`max_accepting`	Number of processes that should accept new connections. Only this many traffic manager child processes will listen for new connections at any one time. Setting this to `0` (zero) will cause your traffic manager to select an appropriate default value based on the architecture and number of CPUs. Value type: `UInt` Default: `<none>`
`idle_connections_max`	The maximum number of unused HTTP keepalive connections with back-end nodes that the traffic manager should maintain for re-use. Setting this to `0` (zero) will cause the traffic manager to auto-size this parameter based on the available number of file-descriptors. Value type: `UInt` Default: `<none>`
`multiple_accept`	Whether or not the traffic manager should try to read multiple new connections each time a new client connects. This can improve performance under some very specific conditions. However, in general it is recommended that this be set to 'false'. Value type: `Boolean` Default: `false`
Configuration keys for the `dns` section:	

`max_ttl`	Maximum Time To Live (expiry time) for entries in the DNS cache. Value type: `UInt` Default: `86400`
`min_ttl`	Minimum Time To Live (expiry time) for entries in the DNS cache. Value type: `UInt` Default: `86400`
`negative_expiry`	Expiry time for failed lookups in the DNS cache. Value type: `UInt` Default: `60`
`size`	Maximum number of entries in the DNS cache. Value type: `UInt` Default: `10867`
`timeout`	Timeout for receiving a response from a DNS server. Value type: `UInt` Default: `12`
Configuration keys for the `ec2` section:	
`access_key_id`	Amazon EC2 Access Key ID. Value type: `String` Default: `<none>`
`secret_access_key`	Amazon EC2 Secret Access Key. Value type: `Password` Default: `<none>`
Configuration keys for the `eventing` section:	
`mail_interval`	The minimum length of time that must elapse between alert emails being sent. Where multiple alerts occur inside this timeframe, they will be

<table>
<tr><td></td><td>retained and sent within a single email rather than separately.

Value type: <code>UInt</code>

Default: <code>30</code></td></tr>
<tr><td><code>max_attempts</code></td><td>The number of times to attempt to send an alert email before giving up.

Value type: <code>UInt</code>

Default: <code>10</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>fault_tolerance</code> section:</td></tr>
<tr><td><code>arp_count</code></td><td>The number of ARP packets a traffic manager should send when an IP address is raised.

Value type: <code>UInt</code>

Default: <code>10</code></td></tr>
<tr><td><code>auto_failback</code></td><td>Whether or not traffic IPs automatically move back to machines that have recovered from a failure and have dropped their traffic IPs.

Value type: <code>Boolean</code>

Default: <code>true</code></td></tr>
<tr><td><code>frontend_check_ips</code></td><td>The IP addresses used to check front-end connectivity. Set this to an empty string if the traffic manager is on an Intranet with no external connectivity.

Value type: <code>Set(String)</code>

Default: <code>%gateway%</code></td></tr>
<tr><td><code>heartbeat_method</code></td><td>The method traffic managers should use to exchange cluster heartbeat messages.

Value type: <code>Enum(String)</code>

Default: <code>unicast</code>

Permitted values:<table><tr><td><code>multicast</code></td><td>multicast</td></tr><tr><td><code>unicast</code></td><td>unicast</td></tr></table></td></tr>
<tr><td><code>monitor_interval</code></td><td>The frequency, in milliseconds, that each traffic manager machine should check and announce its connectivity.</td></tr>
</table>

	Value type: `UInt` Default: `500`
`monitor_timeout`	How long, in seconds, each traffic manager should wait for a response from its connectivity tests or from other traffic manager machines before registering a failure. Value type: `UInt` Default: `5`
`multicast_address`	The multicast address and port to use to exchange cluster heartbeat messages. Value type: `String` Default: `239.100.1.1:9090`
`unicast_port`	The unicast UDP port to use to exchange cluster heartbeat messages. Value type: `UInt` Default: `9090`
`use_bind_ip`	Whether or not cluster heartbeat messages should only be sent and received over the management network. Value type: `Boolean` Default: `false`
`verbose`	Whether or not a traffic manager should log all connectivity tests. This is very verbose, and should only be used for diagnostic purposes. Value type: `Boolean` Default: `false`
Configuration keys for the `ftp` section:	
`data_bind_low`	Whether or not the traffic manager should permit use of FTP data connection source ports lower than 1024. If `No` the traffic manager can completely drop root privileges, if `Yes` some or all privileges may be retained in order to bind to low ports. Value type: `Boolean`

	Default: `false`
Configuration keys for the `glb` section:	
`verbose`	Write a message to the logs for every DNS query that is load balanced, showing the source IP address and the chosen datacenter. Value type: `Boolean` Default: `false`
Configuration keys for the `historical_activity` section:	
`keep_days`	Number of days to store historical traffic information, if set to `0` the data will be kept indefinitely. Value type: `UInt` Default: `90`
Configuration keys for the `java` section:	
`classpath`	CLASSPATH to use when starting the Java runner. Value type: `String` Default: `<none>`
`command`	Java command to use when starting the Java runner, including any additional options. Value type: `String` Default: `java -server`
`enabled`	Whether or not Java support should be enabled. If this is set to `No`, then your traffic manager will not start any Java processes. Java support is only required if you are using the TrafficScript `java.run()` function. Value type: `Boolean` Default: `true`
`lib`	Java library directory for additional jar files. The Java runner will load classes from any `.jar` files stored in this directory, as well as the * jar files and classes stored in traffic manager's catalog. Value type: `String`

<table>
<tr><td></td><td>Default: <code><none></code></td></tr>
<tr><td><code>max_connections</code></td><td>Maximum number of simultaneous Java requests. If there are more than this many requests, then further requests will be queued until the earlier requests are completed. This setting is per-CPU, so if your traffic manager is running on a machine with 4 CPU cores, then each core can make this many requests at one time.

Value type: <code>UInt</code>

Default: <code>256</code></td></tr>
<tr><td><code>session_age</code></td><td>Default time to keep a Java session.

Value type: <code>UInt</code>

Default: <code>86400</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>log</code> section:</td></tr>
<tr><td><code>error_level</code></td><td>The minimum severity of events/alerts that should be logged to disk. <code>INFO</code> will log all events; a higher severity setting will log fewer events. More fine-grained control can be achieved using events and actions.

Value type: <code>Enum(UInt)</code>

Default: <code>info</code>

Permitted values:
<table>
<tr><td><code>fatal</code></td><td>Only fatal errors are logged</td></tr>
<tr><td><code>info</code></td><td>All events are logged to disk</td></tr>
<tr><td><code>serious</code></td><td>Only serious errors or worse</td></tr>
<tr><td><code>warn</code></td><td>Only warnings and errors are logged</td></tr>
</table></td></tr>
<tr><td><code>log_file</code></td><td>The file to log event messages to.

Value type: <code>String</code>

Default: <code>%zeushome%/zxtm/log/errors</code></td></tr>
<tr><td><code>flush_time</code></td><td>How long to wait before flushing the request log files for each virtual server

Value type: <code>UInt</code>

Default: <code>5</code></td></tr>
</table>

`rate`	The maximum number of connection errors logged per second when connection error reporting is enabled. Value type: `UInt` Default: `50`
`reopen`	How long to wait before re-opening request log files, this ensures that log files will be recreated in the case of log rotation. Value type: `UInt` Default: `30`
`time`	The minimum time between log messages for log intensive features such as SLM. Value type: `UInt` Default: `60`
Configuration keys for the `recent_connections` section:	
`max_per_process`	How many recently closed connections each traffic manager process should save. These saved connections will be shown alongside currently active connections when viewing the Connections page. You should set this value to `0` in a benchmarking or performance-critical environment. Value type: `UInt` Default: `<none>`
`retain_time`	The amount of time for which snapshots will be retained on the Connections page. Value type: `UInt` Default: `60`
`snapshot_size`	The maximum number of connections each traffic manager process should show when viewing a snapshot on the Connections page. This value includes both currently active connections and saved connections. If set to `0` all active and saved connection will be displayed on the Connections page. Value type: `UInt` Default: `500`

Configuration keys for the `rest_api` section:	
`auth_timeout`	Timeout for the REST Authentication cache. Each REST request is supplied with a user and password as there is no concept of a session in REST. These login credentials must be validated each time, but to save requesting repeated external authentications for the same user (from the same IP address) a cache of recent authentications is kept. This timeout is the maximum time a given user can stay in the cache. A setting of 0 disables the cache, forcing every REST request to be authenticated which will affect performance. Value type: `UInt` Default: `120`
`enabled`	Whether or not the REST service is enabled Value type: `Boolean` Default: `false`
`http_max_header_length`	The maximum allowed length in bytes of a HTTP request's headers. Value type: `UInt` Default: `4096`
`replicate_absolute`	Absolute timeout before configuration replication via REST. The time before configuration replication via REST will start, regardless of activity of the REST API. If the REST API is busy, after this time, configuration replication will start. Value type: `UInt` Default: `20`
`replicate_lull`	Lull time for configuration replication via REST. The time of inactivity via the REST API before configuration replication will start. Increasing this value will delay configuration replication among a cluster. Value type: `UInt` Default: `5`
`replicate_timeout`	Configuration replication timeout via REST. The time allowed for configuration replication. On system with slow cluster communications or large configurations, increasing this value will improve replication reliability.

	Value type: `UInt` Default: `10`
Configuration keys for the `security` section:	
`password_allow_consecutive_chars`	Whether or not to allow the same character to appear consecutively in passwords. Value type: `Boolean` Default: `true`
`login_banner_accept`	Whether or not users must explicitly agree to the displayed `login_banner` text before logging in to the Admin Server. Value type: `Boolean` Default: `false`
`login_banner`	Banner text displayed on the Admin Server login page and before logging in to appliance SSH servers. Value type: `FreeformString` Default: `<none>`
`login_delay`	The number of seconds before another login attempt can be made after a failed attempt. Value type: `UInt` Default: `<none>`
`max_login_attempts`	The number of sequential failed login attempts that will cause a user account to be suspended. Setting this to `0` disables this feature. To apply this to users who have never successfully logged in, `track_unknown_users` must also be enabled. Value type: `UInt` Default: `<none>`
`max_login_external`	Whether or not usernames blocked due to the `max_login_attempts` limit should also be blocked from authentication against external services (such as LDAP and RADIUS). Value type: `Boolean`

	Default: `false`
`max_login_suspension_time`	The number of minutes to suspend users who have exceeded the `max_login_attempts` limit. Value type: `UInt` Default: `15`
`password_min_alpha_chars`	Minimum number of alphabetic characters a password must contain. Set to 0 to disable this restriction. Value type: `UInt` Default: `<none>`
`password_min_numeric_chars`	Minimum number of numeric characters a password must contain. Set to `0` to disable this restriction. Value type: `UInt` Default: `<none>`
`password_min_length`	Minimum number of characters a password must contain. Set to `0` to disable this restriction. Value type: `UInt` Default: `<none>`
`password_min_special_chars`	Minimum number of special (non-alphanumeric) characters a password must contain. Set to `0` to disable this restriction. Value type: `UInt` Default: `<none>`
`password_min_uppercase_chars`	Minimum number of uppercase characters a password must contain. Set to `0` to disable this restriction. Value type: `UInt` Default: `<none>`
`password_changes_per_day`	The maximum number of times a password can be changed in a 24-hour period. Set to `0` to disable this restriction. Value type: `UInt`

	Default: `<none>`
`password_reuse_after`	The number of times a password must have been changed before it can be reused. Set to `0` to disable this restriction. Value type: `UInt` Default: `<none>`
`post_login_banner`	Banner text to be displayed on the appliance console after login. Value type: `String` Default: `<none>`
`track_unknown_users`	Whether to remember past login attempts from usernames that are not known to exist (should be set to false for an Admin Server accessible from the public Internet). This does not affect the audit log. Value type: `Boolean` Default: `false`
`ui_page_banner`	Banner text to be displayed on all Admin Server pages. Value type: `String` Default: `<none>`
Configuration keys for the `session` section:	
`asp_cache_size`	The maximum number of entries in the ASP session cache. This is used for storing session mappings for ASP session persistence. Approximately 100 bytes will be pre-allocated per entry. Value type: `UInt` Default: `2048`
`ip_cache_size`	The maximum number of entries in the IP session cache. This is used to provide session persistence based on the source IP address. Approximately 100 bytes will be pre-allocated per entry. Value type: `UInt` Default: `2048`
`j2ee_cache_size`	The maximum number of entries in the J2EE session cache. This is used for storing session mappings for J2EE session persistence.

	Approximately 100 bytes will be pre-allocated per entry. Value type: `UInt` Default: `2048`
`ssl_cache_size`	The maximum number of entries in the SSL session persistence cache. This is used to provide session persistence based on the SSL session ID. Approximately 200 bytes will be pre-allocated per entry. Value type: `UInt` Default: `2048`
`universal_cache_size`	The maximum number of entries in the global universal session cache. This is used for storing session mappings for universal session persistence. Approximately 100 bytes will be pre-allocated per entry. Value type: `UInt` Default: `2048`
Configuration keys for the `snmp` section:	
`user_counters`	The number of user defined SNMP counters. Value type: `UInt` Default: `10`
Configuration keys for the `soap` section:	
`idle_minutes`	The number of minutes that the SOAP server should remain idle before exiting. The SOAP server has a short startup delay the first time a SOAP request is made, subsequent SOAP requests don't have this delay. Value type: `UInt` Default: `10`
Configuration keys for the `ssl` section:	
`cache_expiry`	How long the SSL session IDs for SSL decryption should be stored for. Value type: `UInt` Default: `1800`
`cache_size`	How many entries the SSL session ID cache should hold. This cache is used to cache SSL sessions to help speed up SSL handshakes when

	performing SSL decryption. Each entry will allocate approximately 1.2kB of metadata. Value type: `UInt` Default: `6151`
`crl_mem_size`	How much shared memory to allocate for loading Certificate Revocation Lists. This should be at least 3 times the total size of all CRLs on disk. This is specified as either a percentage of system RAM, `1%` for example, or an absolute size such as `10MB`. Value type: `String` Default: `5MB`
`insert_extra_fragment`	Whether or not SSL3 and TLS1 use one-byte fragments as a BEAST countermeasure Value type: `Boolean` Default: `false`
`max_handshake_message_size`	The maximum size (in bytes) of SSL handshake messages that SSL connections will accept. To accept any size of handshake message the key should be set to the value `0`. Value type: `UInt` Default: `10240`
`ocsp_cache_size`	The maximum number of cached OCSP results stored. This cache is used to speed up OCSP checks by caching results. Approximately 1040 bytes are pre-allocated per entry. Value type: `UInt` Default: `2048`
`ssl3_allow_rehandshake`	Whether or not SSL3/TLS re-handshakes should be supported. Enabling support for re-handshakes can expose services to Man-in-the-Middle attacks. It is recommended that only "safe" handshakes be permitted, or none at all. Value type: `Enum(String)` Default: `safe` Permitted values: `always` — Always allow

<table>
<tr><td></td><td><table>
<tr><td>never</td><td>Never allow</td></tr>
<tr><td>rfc5746</td><td>Only if client uses RFC 5746 (Secure Renegotiation Extension)</td></tr>
<tr><td>safe</td><td>Allow safe re-handshakes</td></tr>
</table></td></tr>
<tr><td>ssl3_ciphers</td><td>The SSL ciphers to use. For information on supported ciphers see the online help.
Value type: String
Default: <none></td></tr>
<tr><td>ssl3_diffie_hellman_key_length</td><td>The length in bits of the Diffie-Hellman key for ciphers that use Diffie-Hellman key agreement.
Value type: Enum(UInt)
Default: dh_1024
Permitted values:<table>
<tr><td>dh_1024</td><td>1024</td></tr>
<tr><td>dh_2048</td><td>2048</td></tr>
<tr><td>dh_3072</td><td>3072</td></tr>
<tr><td>dh_4096</td><td>4096</td></tr>
</table></td></tr>
<tr><td>support_ssl2</td><td>Whether or not SSL2 support is enabled.
Value type: Boolean
Default: false</td></tr>
<tr><td>support_ssl3</td><td>Whether or not SSL3 support is enabled.
Value type: Boolean
Default: true</td></tr>
<tr><td>support_tls1</td><td>Whether or not TLS1.0 support is enabled.
Value type: Boolean
Default: true</td></tr>
<tr><td>support_tls1_1</td><td>Whether or not TLS1.1 support is enabled.
Value type: Boolean</td></tr>
</table>

<table>
<tr><td></td><td>Default: <code>true</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>ssl_hardware</code> section:</td></tr>
<tr><td><code>accel</code></td><td>Whether or not the SSL hardware is an "accelerator" (faster than software). By default the traffic manager will only use the SSL hardware if a key requires it (i.e. the key is stored on secure hardware and the traffic manager only has a placeholder/identifier key). With this option enabled, your traffic manager will instead try to use hardware for all SSL decrypts.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>driver_pkcs11_debug</code></td><td>Print verbose information about the PKCS11 hardware security module to the event log.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>driver_pkcs11_lib</code></td><td>The location of the PKCS#11 library for your SSL hardware if it is not in a standard location. The traffic manager will search the standard locations by default.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>driver_pkcs11_slot_desc</code></td><td>The label of the SSL Hardware slot to use. Only required if you have multiple HW accelerator slots.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>driver_pkcs11_slot_type</code></td><td>The type of SSL hardware slot to use.

Value type: <code>Enum(String)</code>

Default: <code>operator</code>

Permitted values:
<table>
<tr><td><code>module</code></td><td>Local Module</td></tr>
<tr><td><code>operator</code></td><td>Operator Card Set</td></tr>
<tr><td><code>softcard</code></td><td>Soft Card</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>driver_pkcs11_user_pin</td><td>The User PIN for the PKCS token (PKCS#11 devices only).

Value type: Password

Default: <none></td></tr>
<tr><td>failure_count</td><td>The number of consecutive failures from the SSL hardware that will be tolerated before the traffic manager assumes its session with the device is invalid and tries to log in again. This is necessary when the device reboots following a power failure.

Value type: UInt

Default: 5</td></tr>
<tr><td>library</td><td>The type of SSL hardware to use. The drivers for the SSL hardware should be installed and accessible to the traffic manager software.

Value type: Enum(String)

Default: none

Permitted values:
<table>
<tr><td>cn1000</td><td>Cavium Networks CN1000</td></tr>
<tr><td>cn2000</td><td>Cavium Networks CN2000</td></tr>
<tr><td>none</td><td>None</td></tr>
<tr><td>pkcs11</td><td>PKCS#11 (e.g. nCipher NetHSM, Sun SCA 6000)</td></tr>
</table>
</td></tr>
<tr><td colspan="2">Configuration keys for the trafficscript section:</td></tr>
<tr><td>array_elements</td><td>The amount of storage that will be allocated to array elements in TrafficScript. If more elements are required then the necessary memory will be allocated during the execution of the rule.

Value type: UInt

Default: 100000</td></tr>
<tr><td>data_size</td><td>The maximum amount of memory available to store TrafficScript data.set() information. This can be specified as a percentage of system RAM, 5% for example; or an absolute size such as 200MB.

Value type: String

Default: 5%</td></tr>
</table>

max_instr	The maximum number of instructions a TrafficScript rule will run. A rule will be aborted if it runs more than this number of instructions without yielding, preventing infinite loops. Value type: UInt Default: 100000
memory_warning	Raise an event if a TrafficScript rule requires more than this amount of buffered network data. If you get such events repeatedly, you may want to consider re-working some of your TrafficScript rules to use less memory or to stream the data that they process rather than storing it all in memory. This setting also limits the amount of data that can be returned by request.GetLine(). Value type: UInt Default: 1048576
regex_cache_size	The maximum number of regular expressions to cache in TrafficScript. Regular expressions will be compiled in order to speed up their use in the future. Value type: UInt Default: 57
regex_match_limit	The maximum number of ways TrafficScript will attempt to match a regular expression at each position in the subject string, before it aborts the rule and reports a TrafficScript error. Value type: UInt Default: 10000000
regex_match_warn_percentage	The percentage of regex_match_limit at which TrafficScript reports a performance warning. Value type: UInt Default: 5
variable_pool_use	Allow the pool.use and pool.select TrafficScript functions to accept variables instead of requiring literal strings. Enabling this feature has the following effects 1. Your traffic manager may no longer be able to know whether a pool is in use. 2. Errors for pools that aren't in use will not be hidden. 3. Some settings displayed for a Pool may not be appropriate for the type

	of traffic being managed. 4. Pool usage information on the pool edit pages and config summary may not be accurate. 5. Monitors will run for all pools (with this option disabled monitors will only run for Pools that are used). Value type: `Boolean` Default: `false`
Configuration keys for the `web_cache` section:	
`avg_path_length`	The estimated average length of the path (including query string) for resources being cached. An amount of memory equal to this figure multiplied by max_file_num will be allocated for storing the paths for cache entries. This setting can be increased if your web site makes extensive use of long URLs. Value type: `UInt` Default: `512`
`disk`	Whether or not to use a disk-backed (typically SSD) cache. If set to `Yes` cached web pages will be stored in a file on disk. This enables the traffic manager to use a cache that is larger than available RAM. The `size` setting should also be adjusted to select a suitable maximum size based on your disk space. Note that the disk caching is optimized for use with SSD storage. Value type: `Boolean` Default: `false`
`disk_dir`	If disk caching is enabled, this sets the directory where the disk cache file will be stored. The traffic manager will create a file called `webcache.data` in this location. Note that the disk caching is optimized for use with SSD storage. Value type: `String` Default: `%zeushome%/zxtm/internal`
`max_file_num`	Maximum number of entries in the cache. Approximately 0.9 KB will be pre-allocated per entry for metadata, this is in addition to the memory reserved for the content cache and for storing the paths of the cached resources. Value type: `UInt`

	Default: `10000`
`max_file_size`	Largest size of a cacheable object in the cache. This is specified as either a percentage of the total cache size, `2%` for example, or an absolute size such as `20MB`. Value type: `String` Default: `2%`
`max_path_length`	The maximum length of the path (including query string) for the resource being cached. If the path exceeds this length then it will not be added to the cache. Value type: `UInt` Default: `2048`
`normalize_query`	Enable normalization (lexical ordering of the parameter-assignments) of the query string. Value type: `Boolean` Default: `true`
`size`	The maximum size of the HTTP web page cache. This is specified as either a percentage of system RAM, `20%` for example, or an absolute size such as `200MB`. Value type: `String` Default: `20%`
`verbose`	Add an X-Cache-Info header to every HTTP response, showing whether the request and/or the response was cacheable. Value type: `Boolean` Default: `false`

SLM Class

Config path: `slm/*`

Service level monitoring is used to produce alerts when an application's performance is degraded. This is done by monitoring the response time of connections to a virtual server.

Key	Description
`note`	A description for the SLM class. Value type: `FreeformString` Default: `<none>`
`response_time`	Responses that arrive within this time limit, expressed in milliseconds, are treated as conforming. Value type: `UInt` Default: `1000`
`serious_threshold`	When the percentage of conforming responses drops below this level, a serious error level message will be emitted. Value type: `UInt` Default: `<none>`
`warning_threshold`	When the percentage of conforming responses drops below this level, a warning message will be emitted. Value type: `UInt` Default: `50`

SSL Trusted Certificate

Config path: `ssl/cas/*`

SSL certificate authority certificates (CAs) and certificate revocation lists (CRLs) can be used when validating server and client certificates.

Key	Description
There are no configuration keys to display for this resource.	

SSL Client Key Pair

Config path: `ssl/client_keys/*`

SSL Client Certificates are used when connecting to backend nodes that require client certificate authentication.

Key	Description
`public`	Public certificate Value type: `FreeformString` Default: `<none>`
`request`	Certificate Signing Request for certificate Value type: `FreeformString` Default: `<none>`
`private`	Private key for certificate Value type: `FreeformString` Default: `<none>`
`note`	Notes for this certificate Value `FreeformString`

	type: Default: `<none>`

SSL Key Pair

Config path: `ssl/server_keys/*`

SSL Server Certificates are presented to clients by virtual servers when SSL decryption is enabled.

Key	Description
`public`	Public certificate Value type: `FreeformString` Default: `<none>`
`request`	Certificate Signing Request for certificate Value type: `FreeformString` Default: `<none>`
`private`	Private key for certificate Value type: `FreeformString` Default: `<none>`
`note`	Notes for this certificate Value type: `FreeformString` Default: `<none>`

Traffic Manager

Config path: `zxtms/*`

Settings that alter the behavior of services running on a single machine.

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td><code>location</code></td><td>This is the location of the local traffic manager is in.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>nameip</code></td><td>Replace Traffic Manager name with an IP address.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>num_children</code></td><td>The number of worker processes the software will run. By default, one child process will be created for each CPU on the system. You may wish to reduce this to effectively "reserve" CPU(s) for other processes running on the host system.

Value type: <code>UInt</code>

Default: <code><none></code></td></tr>
<tr><td><code>trafficip</code></td><td>A table mapping interfaces to networks, used by the traffic manager to select which interface to raise a Traffic IP on.

primary key:<table><tr><td><code>name (String)</code></td><td>A network interface.</td></tr></table>
sub keys:<table><tr><td><code>networks (Set(String))</code></td><td>A set of IP/masks to which the network interface maps.</td></tr></table></td></tr>
<tr><td colspan="2">Configuration keys for the <code>appliance</code> section:</td></tr>
<tr><td><code>gateway_ipv4</code></td><td>The default gateway.

Value type: <code>String</code></td></tr>
</table>

	Default: `<none>`
`gateway_ipv6`	The default IPv6 gateway. Value type: `String` Default: `<none>`
`hostname`	Name (hostname.domainname) of the appliance. Value type: `String` Default: `<none>`
`licence_agreed`	Whether or not the license agreement has been accepted. This determines whether or not the initial configuration (startup) wizard is displayed. Value type: `Boolean` Default: `false`
`name_servers`	The IP addresses of the nameservers the appliance should use and place in `/etc/resolv.conf`. Value type: `String` Default: `<none>`
`ncss_nethsm`	The IP address of the nCipher NetHSM that the appliance should use. Value type: `String` Default: `<none>`
`ncss_nethsm_esn`	The electronic serial number (ESN) for the configured NetHSM. Value type: `String` Default: `<none>`

`ncss_nethsm_hash`	The key hash for the configured NetHSM. Value type: `String` Default: `<none>`
`ncss_rfs`	The IP address of the nCipher Remote File System that the appliance should use. Value type: `String` Default: `<none>`

`ntpservers`	The NTP servers the appliance should use to synchronize its clock.
	
	Value type: `String`
	
	Default:
	`0.riverbed.pool.ntp.org`
	`1.riverbed.pool.ntp.org`
	`2.riverbed.pool.ntp.org`
	`3.riverbed.pool.ntp.org`

`search_domains`	The search domains the appliance should use and place in `/etc/resolv.conf`. Value type: `String` Default: `<none>`
`shim_client_id`	The client ID provided by the portal for this server. Value type: `String` Default: `<none>`
`shim_client_key`	The client key provided by the portal for this server. Value type: `String` Default: `<none>`
`shim_enabled`	Enable the Cloud Steelhead discovery agent on this appliance.

<table>
<tr><td></td><td>Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>shim_ips</code></td><td>The IP addresses of the Cloud Steelheads to use, as a space or comma separated list. If using priority load balancing this should be in ascending order of priority (highest priority last).

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>shim_load_balance</code></td><td>The load balancing method for the selecting a Cloud Steelhead appliance.

Value type: <code>Enum(String)</code>

Default: <code>round_robin</code>

Permitted values:<table><tr><td><code>priority</code></td><td>Priority</td></tr><tr><td><code>round_robin</code></td><td>Round Robin</td></tr></table></td></tr>
<tr><td><code>shim_log_level</code></td><td>The minimum severity that the discovery agent will record to its log.

Value type: <code>Enum(UInt)</code>

Default: <code>notice</code>

Permitted values:<table><tr><td><code>critical</code></td><td>Log critical errors</td></tr><tr><td><code>debug</code></td><td>Log debug or more severe errors (all errors)</td></tr><tr><td><code>info</code></td><td>Log info or more severe errors</td></tr><tr><td><code>notice</code></td><td>Log notice or more severe errors</td></tr><tr><td><code>serious</code></td><td>Log serious or more severe errors</td></tr><tr><td><code>warning</code></td><td>Log warning or more severe errors</td></tr></table></td></tr>
<tr><td><code>shim_mode</code></td><td>The mode used to discover Cloud Steelheads in the local cloud or data center.</td></tr>
</table>

<table>
<tr><td></td><td>Value type: Enum(String)

Default: portal

Permitted values:
<table>
<tr><td>local</td><td>Local Portal</td></tr>
<tr><td>manual</td><td>Manual</td></tr>
<tr><td>portal</td><td>Riverbed Portal</td></tr>
</table>
</td></tr>
<tr><td>shim_portal_url</td><td>The hostname or IP address of the local portal to use.

Value type: String

Default: <none></td></tr>
<tr><td>shim_proxy_host</td><td>The IP or hostname of the proxy server to use to connect to the portal. Leave blank to not use a proxy server.

Value type: String

Default: <none></td></tr>
<tr><td>shim_proxy_port</td><td>The port of the proxy server, must be set if a proxy server has been configured.

Value type: String

Default: <none></td></tr>
<tr><td>ssh_enabled</td><td>Whether or not the SSH server is enabled on the appliance.

Value type: Boolean

Default: true</td></tr>
<tr><td>ssh_port</td><td>The port that the SSH server should listen on.

Value type: UInt

Default: 22</td></tr>
<tr><td>timezone</td><td>The timezone the appliance should use. This must be a path to</td></tr>
</table>

<table>
<tr><td></td><td>a timezone file that exists under <code>/usr/share/zoneinfo/</code>.

Value type: <code>String</code>

Default: <code>US/Pacific</code></td></tr>
<tr><td><code>vlans</code></td><td>The VLANs the software should raise. A VLAN should be configured using the format <code><dev>.<vlanid></code>, where <code><dev></code> is the name of a network device that exists in the host system, <code>eth0.100</code> for example.

Value type: <code>Set(String)</code>

Default: <code><none></code></td></tr>
<tr><td><code>hosts</code></td><td>A table of hostname to static ip address mappings, to be placed in the /etc/hosts file.

primary key:<table><tr><td><code>name (String)</code></td><td>The name of a host.</td></tr></table>
sub keys:<table><tr><td><code>ip_address (String)</code></td><td>The static IP address of the host.</td></tr></table></td></tr>
<tr><td><code>if</code></td><td>A table of network interface specific settings.

primary key:<table><tr><td><code>name (String)</code></td><td>A network interface name.</td></tr></table>
sub keys:<table><tr><td><code>autoneg (Boolean)</code></td><td>Whether auto-negotiation should be enabled for the interface.</td></tr><tr><td><code>bmode (Enum(String))</code></td><td>The trunking mode used for the interface (only 802.3ad is currently supported).

Permitted values:<table><tr><td><code>802_3ad</code></td><td>IEEE 802.3ad</td></tr><tr><td><code>balance_alb</code></td><td>Adaptive Load Balancing</td></tr></table></td></tr><tr><td><code>bond (String)</code></td><td>The trunk of which the interface should be a member.</td></tr><tr><td><code>duplex</code></td><td>Whether full-duplex should be</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>(Boolean)</td><td>enabled for the interface.</td></tr>
<tr><td>mtu (UInt)</td><td>The maximum transmission unit (MTU) of the interface.</td></tr>
<tr><td>speed (Enum(String))</td><td>The speed of the interface.

Permitted values:
<table>
<tr><td>10</td><td>10Mbs</td></tr>
<tr><td>100</td><td>100Mbs</td></tr>
<tr><td>1000</td><td>1Gbs</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>ip</td><td>A table of network interfaces and their network settings.

primary key:
<table>
<tr><td>name (String)</td><td>A network interface name.</td></tr>
</table>
sub keys:
<table>
<tr><td>addr (String)</td><td>The IP address for the interface.</td></tr>
<tr><td>isexternal (Boolean)</td><td>Whether the interface is externally facing.</td></tr>
<tr><td>mask (String)</td><td>The IP mask (netmask) for the interface.</td></tr>
</table>
</td></tr>
<tr><td>routes</td><td>A table of destination IP addresses and routing details to reach them.

primary key:
<table>
<tr><td>name (String)</td><td>A destination IP address.</td></tr>
</table>
sub keys:
<table>
<tr><td>gw (String)</td><td>The gateway IP to configure for the route.</td></tr>
<tr><td>if (String)</td><td>The network interface to configure for the route.</td></tr>
<tr><td>mask (String)</td><td>The netmask to apply to the IP address.</td></tr>
</table>
</td></tr>
<tr><td colspan="2">Configuration keys for the cluster_comms section:</td></tr>
<tr><td>bind_ip</td><td>The IP address that the software should bind to for internal administration communications. See also port. If the software is not part of a cluster the default is to use 127.0.0.1 and there should be no reason to touch this setting. If the software is part of a cluster then the default is to listen on all raised IPs, in this case an alternative configuration is to listen on a single</td></tr>
</table>

	IP address. This may be useful if you have a separate management network and wish to restrict control messages to it. It is important to ensure that the `allowed_update_hosts` (in the `Global Settings` resource) is compatible with the IP configured here. Value type: `String` Default: `*`
`allow_update`	Whether or not this instance of the software can send configuration updates to other members of the cluster. When not clustered this key is ignored. When clustered the value can only be changed by another machine in the cluster that has `allow_update` set to `true`. If set to `false` then it will not be possible to log into the admin server for this instance. Value type: `Boolean` Default: `true`
`port`	The port that the software should listen on for internal administration communications. See also `bind_ip`. Value type: `UInt` Default: `9080`
`external_ip`	This is the optional external ip of the traffic manager, which is used to circumvent natting when traffic managers in a cluster span different networks. Value type: `String` Default: `<none>`
Configuration keys for the `java` section:	
`port`	The port the Java Extension handler process should listen on. This port will be bound for localhost communications only. Value type: `UInt` Default: `9060`

Configuration keys for the `rest_api` section:	
`port`	The port that the REST Daemon software should listen on for communications. Value type: `UInt` Default: `9070`
Configuration keys for the `snmp` section:	
`allow`	Restrict which IP addresses can access the SNMP command responder service. The value can be `all`, `localhost`, or a list of IP CIDR subnet masks. For example `10.100.0.0/16` would allow connections from any IP address beginning with `10.100`. Value type: `Set(String)` Default: `all`
`auth_password`	The authentication password. Required (minimum length 8 characters) if `security_level` includes authentication. Value type: `Password` Default: `<none>`
`bind_ip`	The IP address the SNMP service should bind its listen port to. The value `*` (asterisk) means SNMP will listen on all IP addresses. Value type: `String` Default: `*`
`community`	The community string required for SNMPv1 and SNMPv2c commands. (If empty, all SNMPv1 and SNMPv2c commands will be rejected). Value type: `String` Default: `public`

<table>
<tr><td>enabled</td><td>Whether or not the SNMP command responder service should be enabled on this traffic manager.

Value type: Boolean

Default: false</td></tr>
<tr><td>hash_algorithm</td><td>The hash algorithm for authenticated SNMPv3 communications.

Value type: Enum(String)

Default: md5

Permitted values:<table><tr><td>md5</td><td>MD5</td></tr><tr><td>sha1</td><td>SHA-1</td></tr></table></td></tr>
<tr><td>port</td><td>The port the SNMP command responder service should listen on. The value default denotes port 161 if the software is running with root privileges, and 1161 otherwise.

Value type: String

Default: default</td></tr>
<tr><td>priv_password</td><td>The privacy password. Required (minimum length 8 characters) if security_level includes privacy (message encryption).

Value type: Password

Default: <none></td></tr>
<tr><td>security_level</td><td>The security level for SNMPv3 communications.

Value type: Enum(String)

Default: noauthnopriv

Permitted values:<table><tr><td>authnopriv</td><td>Authentication only</td></tr><tr><td>authpriv</td><td>Authentication and Privacy</td></tr><tr><td>noauthnopriv</td><td>No Authentication, No</td></tr></table></td></tr>
</table>

	Privacy
username	The username required for SNMPv3 commands. (If empty, all SNMPv3 commands will be rejected). Value type: String Default: <none>

Traffic IP Group

Config path: flipper/*

Traffic IP groups are sets of IP addresses that are distributed across a cluster for fault tolerance.

Key	Description
enabled	If set to No, the traffic IP group will be disabled and none of the traffic IP addresses will be raised. Value type: Boolean Default: true
hash_source_port	Whether or not the source port should be taken into account when deciding which traffic manager should handle a request. Value type: Boolean Default: false
ipaddresses	The IP addresses that belong to the Traffic IP group. Value type: Set(String) Default: <none>
keeptogether	If set to Yes then all the traffic IPs will be raised on a single traffic manager. By default they're distributed across all

<table>
<tr><td></td><td>active traffic managers in the traffic IP group.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>location</code></td><td>The location in which the Traffic IP group is based.

Value type: <code>Int</code>

Default: <code><none></code></td></tr>
<tr><td><code>machines</code></td><td>The traffic managers that can host the traffic IP group's IP addresses.

Value type: <code>Set(Reference(config-tm))</code>

Default: <code><none></code></td></tr>
<tr><td><code>mode</code></td><td>The method used to distribute traffic IPs across machines in the cluster. If "multihosted" is used then <code>multicast</code> must be set to an appropriate multicast IP address.

Value type: <code>Enum(String)</code>

Default: <code>singlehosted</code>

Permitted values:<table><tr><td><code>ec2elastic</code></td><td>Use an EC2 Elastic IP address.</td></tr><tr><td><code>multihosted</code></td><td>Raise each address on every machine in the group (Multi-Hosted mode) - IPv4 only</td></tr><tr><td><code>singlehosted</code></td><td>Raise each address on a single machine (Single-Hosted mode)</td></tr></table></td></tr>
<tr><td><code>multicast</code></td><td>The multicast IP address used to duplicate traffic to all traffic managers in the group.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
</table>

note	A description of this traffic IP group. Value type: String Default: <none>
slaves	A list of traffic managers that are in 'passive' mode. This means that in a fully working environment, they will not have any traffic IP addresses assigned to them. Value type: Set(Reference(config-tm)) Default: <none>
ip_mapping	A table assigning traffic IP addresses to machines that should host them. Traffic IP addresses not specified in this table will automatically be assigned to a machine. primary key: ip (String) — A traffic IP address (from the ipaddresses property). sub keys: traffic_manager (String) — The name of the traffic manager that should host the IP address.

Rule

Config path: `rules/*`

TrafficScript rules allow traffic inspection and modification.

Key	Description
There are no configuration keys to display for this resource.	

TrafficScript Authenticator

Config path: `authenticators/*`

TrafficScript authenticators define remote authentication services that can be queried via a TrafficScript rule.

Key	Description
`host`	The hostname or IP address of the remote authenticator. Value type: `String` Default: `<none>`
`note`	A description of the authenticator. Value type: `FreeformString` Default: `<none>`
`port`	The port on which the remote authenticator should be contacted. Value type: `UInt` Default: `389`
Configuration keys for the `ldap` section:	
`attributes`	A list of attributes to return from the search. If blank, no attributes will be returned. If set to '*' then all user attributes will be returned. Value type: `Set(String)` Default: `<none>`
`bind_dn`	The distinguished name (DN) of the 'bind' user. The traffic manager will connect to the LDAP server as this user when searching for user records. Value type: `String`

	Default: `<none>`
`bind_password`	The password for the bind user. Value type: `Password` Default: `<none>`
`filter`	The filter used to locate the LDAP record for the user being authenticated. Any occurrences of '`%u`' in the filter will be replaced by the name of the user being authenticated. Value type: `String` Default: `<none>`
`filter_base_dn`	The base distinguished name (DN) under which user records are located on the server. Value type: `String` Default: `<none>`
`ssl_enabled`	Whether or not to enable SSL encryption to the LDAP server. Value type: `Boolean` Default: `false`
`ssl_cert`	The SSL certificate that the traffic manager should use to validate the remote server. If no certificate is specified then no signature validation will be performed. Value type: `Reference(config-ssl-cacrl)` Default: `<none>`
`ssl_type`	The type of LDAP SSL encryption to use. Value type: `Enum(String)` Default: `ldaps`

	Permitted values:	`ldaps`	LDAPS
		`starttls`	Start TLS

User Authenticator

Config path: `auth/*`

A user authenticator is used to allow access to the UI and REST API by querying a remote authentication service.

Key	Description
`description`	A description of the authenticator. Value type: `String` Default: `<none>`
`enabled`	Whether or not this authenticator is enabled. Value type: `Boolean` Default: `false`
`type`	The type and protocol used by this authentication service. Value type: `Enum(String)` Default: `<none>` Permitted values: `ldap` LDAP; `radius` RADIUS; `tacacs_plus` TACACS+
Configuration keys for the `ldap` section:	
`base_dn`	The base DN (Distinguished Name) under which directory searches will be applied. The entries for your users should all appear under this DN. An example of a typical base DN is:

<table>
<tr><td></td><td>OU=users, DC=mycompany, DC=local

Value type: String

Default: <none></td></tr>
<tr><td>bind_dn</td><td>Template to construct the bind DN (Distinguished Name) from the username. The string %u will be replaced by the username. Examples: %u@mycompany.local for Active Directory or cn=%u, dn=mycompany, dn=local for both LDAP and Active Directory.

Value type: String

Default: <none></td></tr>
<tr><td>dn_method</td><td>The bind DN (Distinguished Name) for a user can either be searched for in the directory using the base distinguished name and filter values, or it can be constructed from the username.

Value type: Enum(String)

Default: none

Permitted values:
<table>
<tr><td>construct</td><td>Construct</td></tr>
<tr><td>none</td><td>No setting configured</td></tr>
<tr><td>search</td><td>Search</td></tr>
</table>
</td></tr>
<tr><td>fallback_group</td><td>If the group attribute is not defined, or returns no results for the user logging in, the group named here will be used. If not specified, users will be denied access to the traffic manager if no groups matching a Permission Group can be found for them in the directory.

Value type: String

Default: <none></td></tr>
<tr><td>filter</td><td>A filter that can be used to extract a unique user record located under the base DN (Distinguished Name). The string %u will be replaced by the username. This filter is used to find a user's bind DN when dn_method is set to "Search", and to extract group information if the group filter is not specified. Examples: sAMAccountName=%u for Active</td></tr>
</table>

	Directory, or `uid=%u` for some Unix LDAP schemas. Value type: `String` Default: `<none>`
`group_attribute`	The LDAP attribute that gives a user's group. If there are multiple entries for the attribute all will be extracted and they'll be lexicographically sorted, then the first one to match a Permission Group name will be used. Value type: `String` Default: `<none>`
`group_field`	The sub-field of the group attribute that gives a user's group. For example, if **group_attribute** is `memberOf` and this retrieves values of the form `CN=mygroup, OU=groups, OU=users, DC=mycompany, DC=local` you would set group_field to `CN`. If there are multiple matching fields only the first matching field will be used. Value type: `String` Default: `<none>`
`group_filter`	If the user record returned by **filter** does not contain the required group information you may specify an alternative group search filter here. This will usually be required if you have Unix/POSIX-style user records. If multiple records are returned the list of group names will be extracted from all of them. The string `%u` will be replaced by the username. Example: `(&(memberUid=%u)(objectClass=posixGroup))` Value type: `String` Default: `<none>`
`port`	The port to connect to the LDAP server on. Value type: `UInt` Default: `389`

`search_dn`	The bind DN (Distinguished Name) to use when searching the directory for a user's bind DN. You can leave this blank if it is possible to perform the bind DN search using an anonymous bind. Value type: `String` Default: `<none>`
`search_password`	If binding to the LDAP server using `search_dn` requires a password, enter it here. Value type: `Password` Default: `<none>`
`server`	The IP or hostname of the LDAP server. Value type: `String` Default: `<none>`
`timeout`	Connection timeout in seconds. Value type: `UInt` Default: `30`
Configuration keys for the `radius` section:	
`fallback_group`	If no group is found using the vendor and group identifiers, or the group found is not valid, the group specified here will be used. Value type: `String` Default: `<none>`
`group_attribute`	The RADIUS identifier for the attribute that specifies an account's group. May be left blank if **fallback group** is specified. Value type: `UInt`

	Default: `1`
`group_vendor`	The RADIUS identifier for the vendor of the RADIUS attribute that specifies an account's group. Leave blank if using a standard attribute (i.e. for Filter-Id set group_attribute to 11). Value type: `UInt` Default: `7146`
`nas_identifier`	This value is sent to the RADIUS server. Value type: `String` Default: `<none>`
`nas_ip_address`	This value is sent to the RADIUS server, if left blank the address of the interfaced used to connect to the server will be used. Value type: `String` Default: `<none>`
`port`	The port to connect to the RADIUS server on. Value type: `UInt` Default: `1812`
`secret`	Secret key shared with the RADIUS server. Value type: `Password` Default: `<none>`
`server`	The IP or hostname of the RADIUS server. Value type: `String` Default: `<none>`

<table>
<tr><td><code>timeout</code></td><td>Connection timeout in seconds.

Value type: <code>UInt</code>

Default: <code>30</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>tacacs_plus</code> section:</td></tr>
<tr><td><code>auth_type</code></td><td>Authentication type to use.

Value type: <code>Enum(String)</code>

Default: <code>pap</code>

Permitted values:<table><tr><td><code>ascii</code></td><td>ASCII</td></tr><tr><td><code>pap</code></td><td>PAP</td></tr></table></td></tr>
<tr><td><code>fallback_group</code></td><td>If <code>group_service</code> is not used, or no group value is provided for the user by the TACACS+ server, the group specified here will be used. If this is not specified, users with no TACACS+ defined group will be denied access.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>group_field</code></td><td>The TACACS+ "service" field that provides each user's group.

Value type: <code>String</code>

Default: <code>permission-group</code></td></tr>
<tr><td><code>group_service</code></td><td>The TACACS+ "service" that provides each user's group field.

Value type: <code>String</code>

Default: <code>zeus</code></td></tr>
<tr><td><code>port</code></td><td>The port to connect to the TACACS+ server on.

Value <code>UInt</code></td></tr>
</table>

	type: Default: `49`
`secret`	Secret key shared with the TACACS+ server. Value type: `Password` Default: `<none>`
`server`	The IP or hostname of the TACACS+ server. Value type: `String` Default: `<none>`
`timeout`	Connection timeout in seconds. Value type: `UInt` Default: `30`

User Group

Config path: `groups/*`

Permission groups specify permissions for groups of users. These groups can be given read-write or read-only access to different parts of the configuration hierarchy. Each group will contain a table of permissions. Each table entry has a name that corresponds to a part of the configuration hierarchy, and a corresponding access level. The access level may have values of either `none`, `ro` (read only, this is the default), or `full`. Some permissions have sub-permissions, these are denoted by following the parent permission name with a colon (`:`) followed by the sub-permission name. The built-in `admin` group has a special permission key of `all` with the value `full`, this must not be altered for the `admin` group but can be used in other group configuration files to change the default permission level for the group.

Key	Description
`description`	A description for the group. Value `String`

	type: Default: `<none>`
`password_expire_time`	Members of this group must renew their passwords after this number of days. To disable password expiry for the group set this to `0` (zero). Note that this setting applies only to local users. Value type: `UInt` Default: `<none>`
`timeout`	Inactive UI sessions will timeout after this number of seconds. To disable inactivity timeouts for the group set this to `0` (zero). Value type: `UInt` Default: `30`
`permissions`	A table defining which level of permission this group has for specific configuration elements. primary key: `name (String)` — Configuration element to which this group has a level of permission. sub keys: `access_level (String)` — Permission level for the configuration element (none, ro or full)

Virtual Server

Config path: `vservers/*`

A virtual server represents the front end of a load balanced network service. It processes traffic it receives on a specified port and distributes load over a pool of backend nodes.

Key	Description
`add_cluster_ip`	Whether or not the virtual server should add an "X-Cluster-Client-Ip" header to the request that contains the

	remote client's IP address. Value type: `Boolean` Default: `true`
`bandwidth_class`	The bandwidth management class should this server should use, if any. Value type: `Reference(config-bandwidth)` Default: `<none>`
`connect_timeout`	The time, in seconds, to wait for data from a new connection. If no data is received within this time, the connection will be closed. A value of `0` (zero) will disable the timeout. Value type: `UInt` Default: `10`
`enabled`	Whether the virtual server is enabled Value type: `Boolean` Default: `false`
`ftp_force_server_secure`	Whether or not the virtual server should require that incoming FTP data connections from the nodes originate from the same IP address as the node. Value type: `Boolean` Default: `true`
`glb_services`	Associated GLB services for this virtual server, only applies to when using a DNS Value type: `Set(String)` Default: `<none>`

note	A description for the virtual server. Value type: FreeformString Default: <none>
pool	The default pool to use for traffic. Value type: Reference(config-pool) Default: <none>
port	The port on which to listen for incoming connections. Value type: UInt Default: <none>
protection_class	The service protection class that should be used to protect this server, if any. Value type: String Default: <none>
protocol	The protocol that the virtual server is using. Value type: String Default: http
response_rules	Rules to be applied to responses, in order, comma separated. Value type: List(Reference(config-trafficscript)) Default: <none>
request_rules	Rules to be applied to incoming requests, in order, comma separated. Value List(String)

<table>
<tr><td></td><td>type:

Default: <code><none></code></td></tr>
<tr><td><code>slm_class</code></td><td>The service level monitoring class that this server should use, if any.

Value type: <code>Reference(config-slm)</code>

Default: <code><none></code></td></tr>
<tr><td><code>so_nagle</code></td><td>Whether or not Nagle's algorithm should be used for TCP connections.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>ssl_client_cert_headers</code></td><td>What HTTP headers the virtual server should add to each request to show the data in the client certificate.

Value type: <code>Enum(String)</code>

Default: <code>none</code>

Permitted values:
<table>
<tr><td><code>all</code></td><td>Certificate fields and certificate text</td></tr>
<tr><td><code>none</code></td><td>No data</td></tr>
<tr><td><code>simple</code></td><td>Certificate fields</td></tr>
</table>
</td></tr>
<tr><td><code>ssl_decrypt</code></td><td>Whether or not the virtual server should decrypt incoming SSL traffic.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>listen_on_any</code></td><td>Whether to listen on all IP addresses

Value type: <code>Boolean</code>

Default: <code>true</code></td></tr>
</table>

`listen_on_traffic_ips`	Traffic IP Groups to listen on Value type: `Set(String)` Default: `<none>`
`listen_on_hosts`	Hostnames and IP addresses to listen on Value type: `Set(String)` Default: `<none>`
Configuration keys for the `aptimizer` section:	
`enabled`	Whether the virtual server should aptimize web content Value type: `Boolean` Default: `false`
`profile`	A table of Aptimizer profiles and the application scopes that apply to them. primary key: `name (String)` — The name of an Aptimizer acceleration profile. sub keys: `urls (Set(String))` — The application scopes which apply to the acceleration profile.
Configuration keys for the `connection` section:	
`keepalive`	Whether or not the virtual server should use keepalive connections with the remote clients. Value type: `Boolean` Default: `true`
`keepalive_timeout`	The length of time that the virtual server should keep an idle keepalive connection before discarding it. A value of `0` (zero) will mean that the keepalives are never closed by the traffic manager. Value `UInt`

	type: Default: `10`
`max_client_buffer`	The amount of memory, in bytes, that the virtual server should use to store data sent by the client. Larger values will use more memory, but will minimise the number of `read()` and `write()` system calls that the traffic manager must perform. Value type: `UInt` Default: `65536`
`max_server_buffer`	The amount of memory, in bytes, that the virtual server should use to store data returned by the server. Larger values will use more memory, but will minimise the number of `read()` and `write()` system calls that the traffic manager must perform. Value type: `UInt` Default: `65536`
`server_first_banner`	If specified, the traffic manager will use the value as the banner to send for server-first protocols such as POP, SMTP and IMAP. This allows rules to use the first part of the client data (such as the username) to select a pool. Value type: `String` Default: `<none>`
`timeout`	A connection should be closed if no additional data has been received for this period of time. A value of `0` (zero) will disable this timeout. Value type: `UInt` Default: `300`
Configuration keys for the `connection_errors` section:	
`error_file`	The error message to be sent to the client when the traffic manager detects an internal or backend error for the

<table>
<tr><td></td><td>virtual server.

Value type: <code>Reference(config-extra-file)</code>

Default: <code>Default</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>cookie</code> section:</td></tr>
<tr><td><code>domain</code></td><td>The way in which the traffic manager should rewrite the domain portion of any cookies set by a back-end web server.

Value type: <code>Enum(UInt)</code>

Default: <code>no_rewrite</code>

Permitted values:
<table>
<tr><td><code>no_rewrite</code></td><td>Do not rewrite the domain</td></tr>
<tr><td><code>set_to_named</code></td><td>Rewrite the domain to the named domain value</td></tr>
<tr><td><code>set_to_request</code></td><td>Rewrite the domain to the host header of the request</td></tr>
</table>
</td></tr>
<tr><td><code>new_domain</code></td><td>The domain to use when rewriting a cookie's domain to a named value.

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>path_regex</code></td><td>If you wish to rewrite the path portion of any cookies set by a back-end web server, provide a regular expression to match the path:

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>path_replace</code></td><td>If cookie path regular expression matches, it will be replaced by this substitution. Parameters $1-$9 can be used to represent bracketed parts of the regular expression.</td></tr>
</table>

<table>
<tr><td></td><td>Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>secure</code></td><td>Whether or not the traffic manager should modify the "secure" tag of any cookies set by a back-end web server.

Value type: <code>Enum(UInt)</code>

Default: <code>no_modify</code>

Permitted values:<table><tr><td><code>no_modify</code></td><td>Do not modify the 'secure' tag</td></tr><tr><td><code>set_secure</code></td><td>Set the 'secure' tag</td></tr><tr><td><code>unset_secure</code></td><td>Unset the 'secure' tag</td></tr></table></td></tr>
<tr><td colspan="2">Configuration keys for the <code>ftp</code> section:</td></tr>
<tr><td><code>ssl_data</code></td><td>Use SSL on the data connection as well as the control connection (if not enabled it is left to the client and server to negotiate this).

Value type: <code>Boolean</code>

Default: <code>true</code></td></tr>
<tr><td><code>data_source_port</code></td><td>The source port to be used for active-mode FTP data connections. If 0, a random high port will be used, otherwise the specified port will be used. If a port below 1024 is required you must first explicitly permit use of low ports with the <code>data_bind_low</code> global setting.

Value type: <code>UInt</code>

Default: <code><none></code></td></tr>
<tr><td><code>force_client_secure</code></td><td>Whether or not the virtual server should require that incoming FTP data connections from the client originate from the same IP address as the corresponding client control connection.

Value type: <code>Boolean</code></td></tr>
</table>

	Default: `true`
`port_range_high`	If non-zero, then this controls the upper bound of the port range to use for FTP data connections. Value type: `UInt` Default: `<none>`
`port_range_low`	If non-zero, then this controls the lower bound of the port range to use for FTP data connections. Value type: `UInt` Default: `<none>`
Configuration keys for the `gzip` section:	
`compress_level`	Compression level (1-9, 1=low, 9=high) Value type: `UInt` Default: `1`
`enabled`	Compress web pages sent back by the server Value type: `Boolean` Default: `false`
`include_mime`	MIME types to compress. Complete MIME types can be used, or a type can end in a '*' to match multiple types. Value type: `Set(String)` Default: `text/html text/plain`
`max_size`	Maximum document size to compress (0 means unlimited) Value type: `UInt` Default: `10000000`

<table>
<tr><td>min_size</td><td>Minimum document size to compress

Value type: UInt

Default: 1000</td></tr>
<tr><td>no_size</td><td>Compress documents with no given size

Value type: Boolean

Default: true</td></tr>
<tr><td colspan="2">Configuration keys for the http section:</td></tr>
<tr><td>chunk_overhead_forwarding</td><td>Handling of HTTP chunk overhead. When Stingray receives data from a server or client that consists purely of protocol overhead (contains no payload), forwarding of such segments is delayed until useful payload data arrives (setting "lazy"). Changing this key to "eager" will make Stingray incur the overhead of immediately passing such data on; it should only be used with HTTP peers whose chunk handling requires it.

Value type: Enum(String)

Default: lazy

Permitted values:<table><tr><td>eager</td><td>Forward all data, even when no new payload information is available.</td></tr><tr><td>lazy</td><td>Only forward segments when useful payload data is available.</td></tr></table></td></tr>
<tr><td>location_regex</td><td>If the 'Location' header matches this regular expression, rewrite the header using the 'location_replace' pattern.

Value type: String

Default: <none></td></tr>
<tr><td>location_replace</td><td>If the 'Location' header matches the 'location_regex' regular expression, rewrite the header with this pattern (parameters such as $1-$9 can be used to match parts of</td></tr>
</table>

<table>
<tr><td></td><td>the regular expression):

Value type: <code>String</code>

Default: <code><none></code></td></tr>
<tr><td><code>location_rewrite</code></td><td>The action the virtual server should take if the "Location" header does not match the <code>location_regex</code> regular expression.

Value type: <code>Enum(UInt)</code>

Default: <code>if_host_matches</code>

Permitted values:
<table>
<tr><td><code>always</code></td><td>Rewrite the hostname to the request's "Host" header, and rewrite the protocol and port if necessary;</td></tr>
<tr><td><code>if_host_matches</code></td><td>Do not rewrite the hostname. Rewrite the protocol and port if the hostname matches the request's "Host" header.</td></tr>
<tr><td><code>never</code></td><td>Nothing;</td></tr>
</table>
</td></tr>
<tr><td><code>mime_default</code></td><td>Auto-correct MIME types if the server sends the "default" MIME type for files.

Value type: <code>String</code>

Default: <code>text/plain</code></td></tr>
<tr><td><code>mime_detect</code></td><td>Auto-detect MIME types if the server does not provide them.

Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td colspan="2">Configuration keys for the <code>log</code> section:</td></tr>
</table>

client_connection_failures	Should the virtual server log failures occurring on connections to clients. Value type: Boolean Default: false
enabled	Whether or not to log connections to the virtual server to a disk on the file system. Value type: Boolean Default: false
filename	The name of the file in which to store the request logs. Appliances will ignore this. The filename can contain macros which will be expanded by the traffic manager to generate the full filename. Value type: String Default: %zeushome%/zxtm/log/%v.log
format	The log file format. This specifies the line of text that will be written to the log file when a connection to the traffic manager is completed. Many parameters from the connection can be recorded using macros. Value type: String Default: %h %l %u %t "%r" %s %b "%{Referer}i" "%{User-agent}i"
server_connection_failures	Should the virtual server log failures occurring on connections to nodes. Value type: Boolean Default: false
ssl_failures	Should the virtual server log failures occurring on SSL secure negotiation. Value Boolean

	type: Default: `false`
Configuration keys for the `request_tracing` section:	
`enabled`	Record a trace of major connection processing events for each request and response Value type: `Boolean` Default: `false`
`trace_io`	Include details of individual I/O events in request and response traces. Requires request tracing to be enabled. Value type: `Boolean` Default: `false`
Configuration keys for the `rtsp` section:	
`streaming_port_range_high`	If non-zero this controls the upper bound of the port range to use for streaming data connections. Value type: `UInt` Default: `<none>`
`streaming_port_range_low`	If non-zero this controls the lower bound of the port range to use for streaming data connections. Value type: `UInt` Default: `<none>`
`streaming_timeout`	If non-zero data-streams associated with RTSP connections will timeout if no data is transmitted for this many seconds. Value type: `UInt` Default: `30`

Configuration keys for the `sip` section:	
`dangerous_requests`	The action to take when a SIP request with body data arrives that should be routed to an external IP. Value type: `Enum(String)` Default: `node` Permitted values: `forbid` — Send a 403 Forbidden response to the client `forward` — Forward the request to its target URI (dangerous) `node` — Send the request to a back-end node
`follow_route`	Should the virtual server follow routing information contained in SIP requests. If set to `No` requests will be routed to the chosen back-end node regardless of their URI or Route header. Value type: `Boolean` Default: `true`
`max_connection_mem`	SIP clients can have several pending requests at one time. To protect the traffic manager against DoS attacks, this setting limits the amount of memory each client can use. When the limit is reached new requests will be sent a 413 response. If the value is set to `0` (zero) the memory limit is disabled. Value type: `UInt` Default: `65536`
`mode`	The mode that this SIP virtual server should operate in. Value type: `Enum(String)` Default: `sip_gateway` Permitted values: `full_gateway` — All SIP requests and responses and all session

			data will pass through Stingray. A port range to use for the session data and a timeout value for inactive data connections can be specified in the additional settings that are displayed when the `Full Gateway` mode is selected.
		`route`	The first SIP request in a session will pass through Stingray, along with its responses, but all future requests that are part of the same session will go directly to the back-end node that was chosen by the traffic manager.
		`sip_gateway`	All SIP requests and responses will pass through the traffic manager.
`rewrite_uri`	Replace the Request-URI of SIP requests with the address of the selected back-end node. Value type: `Boolean` Default: `false`		
`streaming_port_range_high`	If non-zero this controls the upper bound of the port range to use for streaming data connections. Value type: `UInt` Default: `<none>`		
`streaming_port_range_low`	If non-zero, then this controls the lower bound of the port range to use for streaming data connections. Value type: `UInt` Default: `<none>`		

`streaming_timeout`	If non-zero a UDP stream will timeout when no data has been seen within this time. Value type: `UInt` Default: `60`
`timeout_messages`	When timing out a SIP transaction, send a 'timed out' response to the client and, in the case of an INVITE transaction, a CANCEL request to the server. Value type: `Boolean` Default: `true`
`transaction_timeout`	The virtual server should discard a SIP transaction when no further messages have been seen within this time. Value type: `UInt` Default: `30`
Configuration keys for the `smtp` section:	
`expect_starttls`	Whether or not the traffic manager should expect the connection to start off in plain text and then upgrade to SSL using STARTTLS when handling SMTP traffic. Value type: `Boolean` Default: `true`
Configuration keys for the `ssl` section:	
`client_cert_cas`	The certificate authorities that this virtual server should trust to validate client certificates. If no certificate authorities are selected, and client certificates are requested, then all client certificates will be accepted. Value type: `Set(String)` Default: `<none>`

<table>
<tr><td>issued_certs_never_expire</td><td>When the virtual server verifies certificates signed by these certificate authorities, it doesn't check the 'not after' date, i.e., they are considered valid even after their expiration date has passed (but not if they have been revoked).

Value type: Set(String)

Default: <none></td></tr>
<tr><td>request_client_cert</td><td>Whether or not the virtual server should request an identifying SSL certificate from each client.

Value type: Enum(UInt)

Default: dont_request

Permitted values:<table><tr><td>dont_request</td><td>Do not request a client certificate</td></tr><tr><td>request</td><td>Request, but do not require a client certificate</td></tr><tr><td>require</td><td>Require a client certificate</td></tr></table></td></tr>
<tr><td>add_http_headers</td><td>Whether or not the virtual server should add HTTP headers to each request to show the SSL connection parameters.

Value type: Boolean

Default: false</td></tr>
<tr><td>ocsp_max_response_age</td><td>The number of seconds for which an OCSP response is considered valid if it has not yet exceeded the time specified in the 'nextUpdate' field. If set to 0 (zero) then OCSP responses are considered valid until the time specified in their 'nextUpdate' field.

Value type: UInt

Default: <none></td></tr>
<tr><td>ocsp_time_tolerance</td><td>The number of seconds outside the permitted range for which the 'thisUpdate' and 'nextUpdate' fields of an OCSP</td></tr>
</table>

	response are still considered valid. Value type: `UInt` Default: `30`
`ocsp_timeout`	The number of seconds after which OCSP requests will be timed out. Value type: `UInt` Default: `10`
`prefer_sslv3`	Whether or not to prefer SSLv3 over TLS when the client appears to support both. SSLv3 is slightly faster, but some clients don't allow SSLv3 but still send the ClientHello inside SSLv2 or SSLv3 records. The default option is to prefer TLS due to known vulnerabilities in the way block ciphers are used before TLSv1.1. Value type: `Boolean` Default: `false`
`send_close_alerts`	Whether or not to send an SSL/TLS "close alert" when the traffic manager is initiating an SSL socket disconnection. Value type: `Boolean` Default: `false`
`trust_magic`	If the traffic manager is receiving traffic sent from another traffic manager, then enabling this option will allow it to decode extra information on the true origin of the SSL connection. This information is supplied by the first traffic manager. Value type: `Boolean` Default: `false`
`ocsp_enable`	Whether or not the traffic manager should use OCSP to check the revocation status of client certificates.

<table>
<tr><td></td><td>Value type: <code>Boolean</code>

Default: <code>false</code></td></tr>
<tr><td><code>ocsp_issuers</code></td><td>A table of certificate issuer specific OCSP settings.

primary key:
<table>
<tr><td><code>issuer (String)</code></td><td>The name of an issuer (or DEFAULT for default OCSP settings).</td></tr>
</table>
sub keys:
<table>
<tr><td><code>aia (Boolean)</code></td><td>Whether the traffic manager should use AIA information contained in a client certificate to determine which OCSP responder to contact.</td></tr>
<tr><td><code>nonce (Enum(String))</code></td><td>How to use the OCSP nonce extension, which protects against OCSP replay attacks. Some OCSP servers do not support nonces.

Permitted values:
<table>
<tr><td><code>off</code></td><td>No nonce check</td></tr>
<tr><td><code>on</code></td><td>Use nonce, server does not have to reply with nonce</td></tr>
<tr><td><code>strict</code></td><td>Use nonce, server must reply with nonce</td></tr>
</table></td></tr>
<tr><td><code>required (Enum(String))</code></td><td>Whether we should do an OCSP check for this issuer, and whether it is required or optional.

Permitted values:
<table>
<tr><td><code>none</code></td><td>None</td></tr>
<tr><td><code>optional</code></td><td>OCSP check</td></tr>
</table></td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td><table><tr><td></td><td>optional</td></tr><tr><td>strict</td><td>OCSP check required</td></tr></table></td></tr>
<tr><td>responder_cert (String)</td><td>The expected responder certificate</td></tr>
<tr><td>signer (String)</td><td>The certificate with which to sign the request, if any.</td></tr>
<tr><td>url (String)</td><td>Which OCSP responders this virtual server should use to verify client certificates.</td></tr>
</table>
</td></tr>
<tr><td>server_cert_default</td><td>The default SSL certificate to use for this virtual server.

Value type: String

Default: <none></td></tr>
<tr><td>server_cert_host_mapping</td><td>Host specific SSL server certificate mappings.

primary key:
<table>
<tr><td>host (String)</td><td>Host which this entry refers to.</td></tr>
</table>
sub keys:
<table>
<tr><td>certificate (String)</td><td>The SSL server certificate for a particular destination site IP.</td></tr>
</table>
</td></tr>
<tr><td colspan="2">Configuration keys for the syslog section:</td></tr>
<tr><td>enabled</td><td>Whether or not to log connections to the virtual server to a remote syslog host.

Value type: Boolean

Default: false</td></tr>
<tr><td>format</td><td>The log format for the remote syslog. This specifies the line of text that will be sent to the remote syslog when a connection to the traffic manager is completed. Many parameters from the connection can be recorded using macros.</td></tr>
</table>

	Value type: `String` Default: `%h %l %u %t "%r" %s %b "%{Referer}i" "%{User-agent}i"`
`ip_end_point`	The remote host and port (default is 514) to send request log lines to. Value type: `String` Default: `<none>`
Configuration keys for the `tcp` section:	
`proxy_close`	If set to `Yes` the traffic manager will send the client FIN to the back-end server and wait for a server response instead of closing the connection immediately. This is only necessary for protocols that require half-close support to function correctly, such as "rsh". If the traffic manager is responding to the request itself, setting this key to Yes will cause the traffic manager to continue writing the response even after it has received a FIN from the client. Value type: `Boolean` Default: `false`
Configuration keys for the `udp` section:	
`end_point_persistence`	Whether or not UDP datagrams from the same IP and port are sent to the same node in the pool if there's an existing UDP transaction. Although it's not always guaranteed as while making a decision to reuse the same node, traffic manager can also apply other protocol specific filtering e.g CallID matching for SIP packets in addition to IP and port matching. Value type: `Boolean` Default: `true`
`port_smp`	Whether or not UDP datagrams should be distributed across all traffic manager processes. This setting is not recommended if the traffic manager will be handling connection-based UDP protocols.

	Value type: `Boolean` Default: `false`
`response_datagrams_expected`	The virtual server should discard any UDP connection and reclaim resources when the node has responded with this number of datagrams. For simple request/response protocols this can be often set to `1`. If set to `-1`, the connection will not be discarded until the `timeout` is reached. Value type: `Int` Default: `1`
`timeout`	The virtual server should discard any UDP connection and reclaim resources when no further UDP traffic has been seen within this time. Value type: `UInt` Default: `7`
Configuration keys for the `web_cache` section:	
`control_out`	The "Cache-Control" header to add to every cached HTTP response, `no-cache` or `max-age=600` for example. Value type: `String` Default: `<none>`
`enabled`	If set to `Yes` the traffic manager will attempt to cache web server responses. Value type: `Boolean` Default: `false`
`error_page_time`	Time period to cache error pages for. Value type: `UInt`

	Default: `30`
`refresh_time`	If a cached page is about to expire within this time, the traffic manager will start to forward some new requests on to the web servers. A maximum of one request per second will be forwarded; the remainder will continue to be served from the cache. This prevents "bursts" of traffic to your web servers when an item expires from the cache. Setting this value to `0` will stop the traffic manager updating the cache before it expires. Value type: `UInt` Default: `2`
`max_time`	Maximum time period to cache web pages for. Value type: `UInt` Default: `600`

CHAPTER 5 Further Information

Stingray Manuals

Your traffic management system includes an **Installation and Getting Started Guide**, intended to get you up and running quickly, and a more detailed **User Manual**. There are also full reference manuals for functionality such as the Java Extensions and TrafficScript.

You can access these manuals via the **Help** pages (described below), or download the most recent versions from the Riverbed Support website at:

https://support.riverbed.com/software/index.htm

Information online

Product specifications can be found at:

http://www.riverbed.com/products-solutions/products/application-delivery-stingray/

Visit the Riverbed Splash community website for further documentation, examples, white papers and other resources:

http://splash.riverbed.com