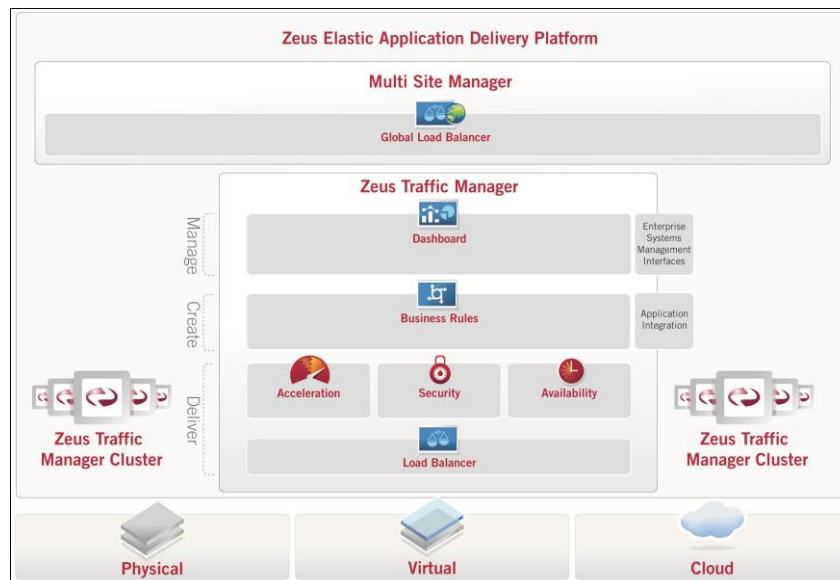


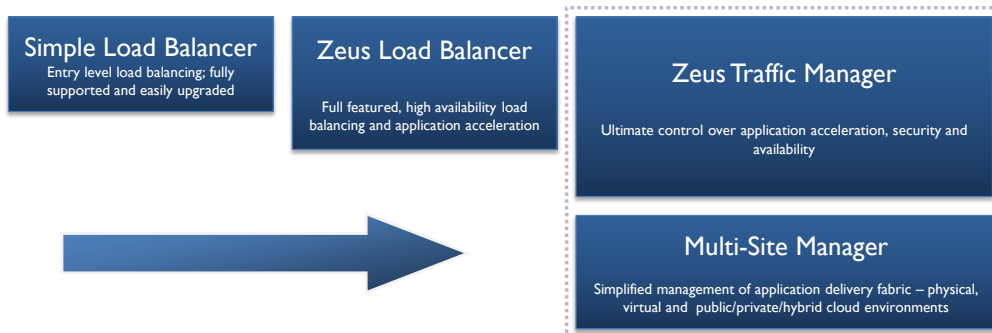
Zeus7: What's new?

Zeus Elastic Application Delivery platform

Zeus' traffic management solutions create an advanced, scalable platform that delivers your applications from any environment – physical, virtual or cloud:



Zeus version 7.0 Traffic Management Products



Zeus Traffic Manager: Application Delivery and Load Balancing software (for Linux and Solaris) and virtual appliance (for VMware, Xen and OracleVM virtualization platforms).

Zeus Load Balancer: Advanced Load Balancing software for smaller organizations who require availability and scalability for their online services.

Zeus Simple Load Balancer: Entry-level, dedicated load balancing solution available through selected cloud and service providers only.

Zeus Multi-Site Manager: Extends Zeus' cluster management to span multiple locations, with an option to load balance traffic globally across those locations.

New Products and features in the Zeus 7.0 release

New product: Zeus Multi-Site Manager

The **Zeus Multi-Site Manager** scales and manages your Zeus Traffic Manager cluster across multiple physical, virtual and cloud locations (new management product for Zeus Traffic Manager and Zeus Load Balancer).

Global Load Balancer module – option for Multi-Site Manager: distributes traffic across multiple locations based on availability, performance, proximity and user-defined rules.

New Traffic Manager modules:

Real-time Analytics module – option for Zeus Traffic Manager: filter and inspect traffic in real-time to determine anomalies and traffic characteristics.

Application Auto-Scaling module – option for Zeus Traffic Manager: based on service performance, automatically initiate scale-up or scale-down actions against supported virtual/cloud infrastructures.

Feature improvements for Zeus Traffic Manager and Zeus Load Balancer:

TrafficScript language improvements – build and manage sophisticated traffic management policies, extending the capabilities and power of Zeus' TrafficScript language with support for arrays, hashes (maps or 'associative arrays'), code libraries and low-level network connections (new feature for Zeus Traffic Manager).

Connection limiting – offloads concurrency to avoid connection timeouts and prevent the degradation in performance caused by overloading fragile servers with too many concurrent connections. Connection limiting respects the internal concurrency limits of applications by queuing transactions within the traffic manager (new feature for Zeus Traffic Manager / Zeus Load Balancer).

Reduced IP usage for fault-tolerance – enabling administrators to create fault-tolerant Zeus clusters without requiring unnecessary external IP addresses (new feature for Zeus Traffic Manager / Zeus Load Balancer).

Fine-grained configuration import and export – easy integration with external configuration management systems by exporting and importing the configuration for an individual service (new feature for Zeus Traffic Manager / Zeus Load Balancer).

Global map visualization – provides a compelling visualization of your site traffic, plotting the source of requests in real-time on a global map (new feature for Zeus Traffic Manager / Zeus Load Balancer);

UI Search – making it easier to find and edit settings when your configuration is large and complex (new feature for Zeus Traffic Manager / Zeus Load Balancer).

Zeus Multi-Site Manager and Global Load Balancing

As a service grows in popularity or sophistication, it is often deployed in multiple locations concurrently. For example, organizations may operate separate development, test and production environments to stage updates to the service, and the production service may be deployed in several geographically dispersed locations for performance and availability reasons.



The **Zeus Multi-Site Manager** will allow you to manage Zeus Traffic Manager clusters across multiple locations. You can control which services run from each cluster, and coordinate the configuration of these services, including configuration settings that are unique or specific to each cluster.

Configuration Locations	Unfold All / Fold All
Locations are groupings of traffic managers that are situated in the same place. Location specific configuration can be added which is only used by traffic managers at that location.	
<p>▼ ✓ New York Used by 1 traffic manager Edit</p> <p>Traffic Managers: zeus1-ny Latitude: 40.77° Longitude: -73.98°</p>	
<p>▼ ✓ San Francisco Used by 1 traffic manager Edit</p> <p>Traffic Managers: zeus1-sf Latitude: 37.75° Longitude: -122.68°</p>	

You can configure locations, and identify where each traffic manager is deployed

Use case – test and production

For example, you could run your primary web site from a test cluster and from multiple production clusters. Multi-Site Manager will control the application delivery configuration in all clusters, respecting local differences such as IP addresses and server nodes. Changes to configuration could be prototyped in the test cluster, then rolled out to production once testing is complete.

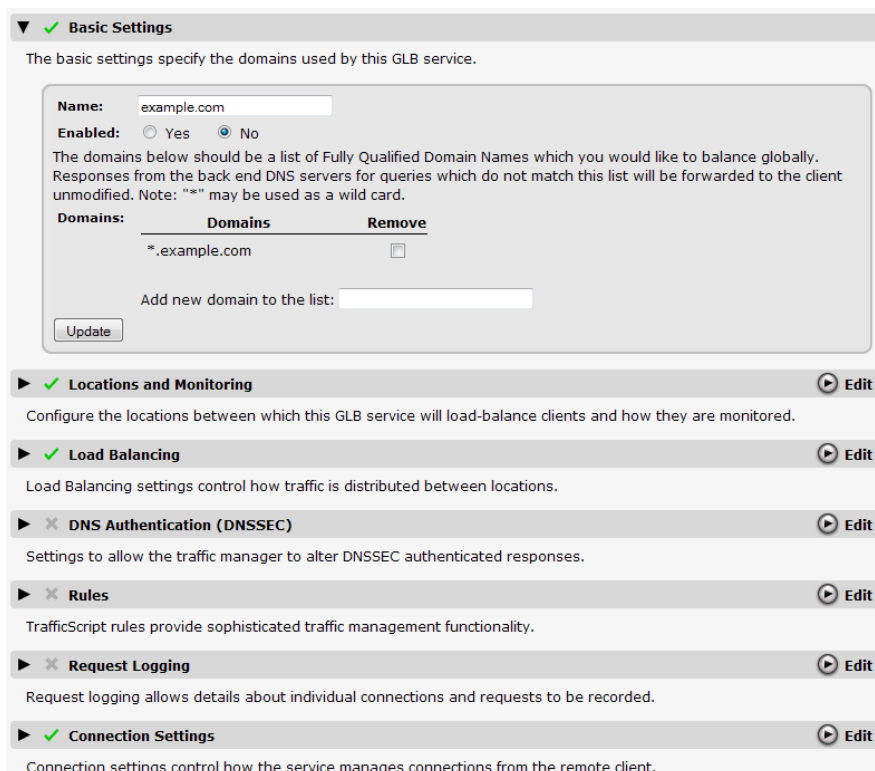
Manage configuration, reporting, visualization and logging across all locations

Multi-Site Manager gives you a global view of your configuration, and identifies cluster-specific differences. It provides detailed reporting and traffic visualization for each cluster and aggregates event logs and diagnostics information so that you have full visibility of your entire application delivery infrastructure from each individual Zeus Administration Server. For maximum reliability, the traffic managers in each cluster are fully fault-tolerant.

Global Load Balancing option

The **Global Load Balancing** option for Multi-Site Manager controls which location each user of a service is directed to. The decision may be based on many criteria, including geographic proximity (for performance reasons), service performance (for load balancing) and availability (for reliability).

Global Load Balancing policies can also be augmented using TrafficScript to give very fine-grained control over site selection. For example, an organization may wish to direct certain internal users to a test location, while all other users are directed to the production instances of a service.



The screenshot shows the configuration interface for Global Load Balancing (GLB) services. It is organized into several sections:

- Basic Settings:** This section is expanded and contains:
 - A description: "The basic settings specify the domains used by this GLB service."
 - A "Name" field with the value "example.com".
 - An "Enabled" section with radio buttons for "Yes" and "No", where "No" is selected.
 - A note: "The domains below should be a list of Fully Qualified Domain Names which you would like to balance globally. Responses from the back end DNS servers for queries which do not match this list will be forwarded to the client unmodified. Note: "*" may be used as a wild card."
 - A "Domains" table with two columns: "Domains" and "Remove". It contains one entry: "*.example.com" with a checkbox in the "Remove" column.
 - An "Add new domain to the list:" text input field.
 - An "Update" button.
- Locations and Monitoring:** A collapsed section with an "Edit" button. Description: "Configure the locations between which this GLB service will load-balance clients and how they are monitored."
- Load Balancing:** A collapsed section with an "Edit" button. Description: "Load Balancing settings control how traffic is distributed between locations."
- DNS Authentication (DNSSEC):** A collapsed section with an "Edit" button. Description: "Settings to allow the traffic manager to alter DNSSEC authenticated responses."
- Rules:** A collapsed section with an "Edit" button. Description: "TrafficScript rules provide sophisticated traffic management functionality."
- Request Logging:** A collapsed section with an "Edit" button. Description: "Request logging allows details about individual connections and requests to be recorded."
- Connection Settings:** A collapsed section with an "Edit" button. Description: "Connection settings control how the service manages connections from the remote client."

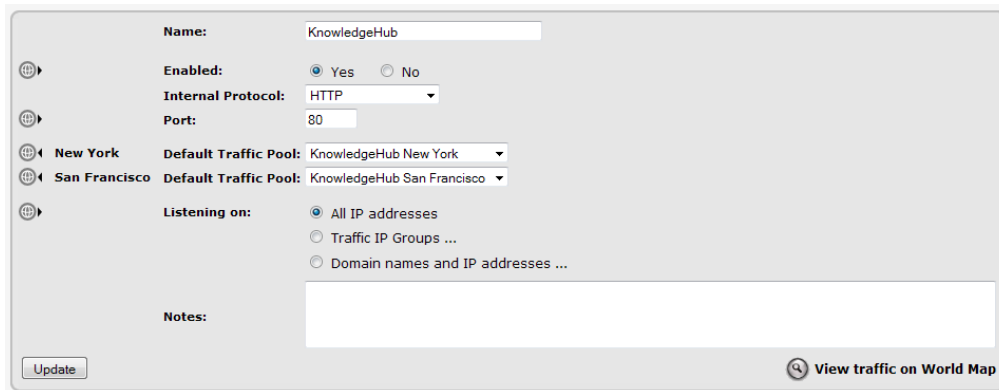
GLB Services are used to configure how the Multi-Site Manager responds to DNS requests

When a service is running from multiple clouds, an organization can control how traffic is distributed to these clouds. This traffic distribution can be used to reduce cloud hosting costs, support cloud bursting, take advantage of price variations and make effective use of reserved instances.

Zeus users - what you need to know

The Multi-Site Manager unites multiple Zeus 'clusters' into a single 'multi-site cluster' that spans multiple locations:

- The Zeus instances in a single cluster are fully fault-tolerant, sharing traffic IP addresses, heartbeats and session persistence information automatically.
- Configuration is shared across all clusters, but becomes location-aware. The same service can have different parameters when running in different cluster; for example, a virtual server may use a different default pool when running from the Paris and San Jose datacenters, and may listen on different traffic IP addresses.



Name:	KnowledgeHub
Enabled:	<input checked="" type="radio"/> Yes <input type="radio"/> No
Internal Protocol:	HTTP
Port:	80
Default Traffic Pool:	KnowledgeHub New York
Default Traffic Pool:	KnowledgeHub San Francisco
Listening on:	<input checked="" type="radio"/> All IP addresses <input type="radio"/> Traffic IP Groups ... <input type="radio"/> Domain names and IP addresses ...
Notes:	

Update View traffic on World Map

Configuration items can have different values in different locations

- Diagnostics, logs and monitoring reports are seamlessly merged so that each Zeus Administration server has a global view of the entire status across all of the clusters.

Global Load Balancing is an optional component of the Multi-Site Manager. When present, it allows you to create special GLB virtual servers that load-balance DNS requests across local or remote DNS servers. These GLB virtual servers can be extended with global load balancing policies that manipulate DNS requests and responses to direct each user to the most appropriate location based on proximity, availability, performance and custom TrafficScript policies.

Global Load Balancing fully supports DNSSEC for DNS response signing. It integrates with Zeus' core traffic management and health checking, and can direct users to external locations where Zeus devices are not present.

All traffic managers managed by the Multi-Site Manager must be running version 7. The Multi-Site manager cannot manage older versions of Zeus Traffic Manager, or clusters with mixed versions.

Real-Time Analytics Module

Denial of service attacks can severely impair the operation of an online service, and intermittent performance or reliability problems can be difficult to identify and manage. Administrators need a real-time view of the traffic they are managing and the effects it has on their service.

The new Real-Time Analytics module is an option for Zeus Traffic Manager. It gives an immediate overview of recent transactions, visualized as a list or on a real-time global map. Administrators can zoom in on the parameters of interest, such as source location or response time and filter the transactions in real time to identify the problem traffic. They can then create TrafficScript rules to control this traffic as necessary.

Use case – denial of service attack

The scenario: an administrator observes that performance is suffering, and the traffic manager is processing many more transactions than is normal.

The investigation: A quick inspection of the global map showing traffic sources shows a large number of connections coming from a particular location. The administrator filters traffic based on these IP addresses and observes that the majority of requests contain randomly generated URLs that are triggering an application on the website.

When inspecting several of these unusual requests, that administrator notices that they perhaps have the wrong host header or an unusual URL. Both situations are typical of an automatic denial of service attack.

The solution: The administrator creates a TrafficScript rule that drops all traffic from the suspect source location if the traffic matches the attack pattern (for example, has an incorrect host header). The real-time analysis confirms the traffic is being dropped, and the current activity monitors demonstrate that the outgoing bandwidth has dropped and performance has recovered.

Use case – overloaded application

The scenario: Users report that the customer self care interface is 'slow' and often returns errors.

The investigation: The administrator looks at the real-time analytics report, filtering to only display connections that took more than 500ms to respond. He observes that a large number of connections to the login page are running slowly and many of these return timeout errors; he concludes that the login page is overloaded, perhaps because of a bottleneck in the authorization service.

The solution: While investigating the root cause of the problem, the administrator uses Zeus Traffic Manager to put a rate limit in place to limit the number of login attempts that each user can make each minute. He also adds a TrafficScript rule to replace the unfriendly 'timeout' server error message with a more appropriate web page.

Zeus users – what you need to know

Real-Time Analytics extends the 'Current Connections' list in the Zeus administration interface. It allows you to specify a range of filters so that you can focus on the connections of interest.

Connections

This table shows a summary of the connections currently being processed and also shows several of the most recent completed requests from each traffic manager in the cluster. The number of connections stored in a snapshot, and the amount of time for which a snapshot is retained can be adjusted on the **Global Settings page**.

Connection Filters

No filters defined, displaying all connections.

Add Filter:

Snapshot taken at 7 Sep 04:35:43 (0 seconds ago, 0 connections since)

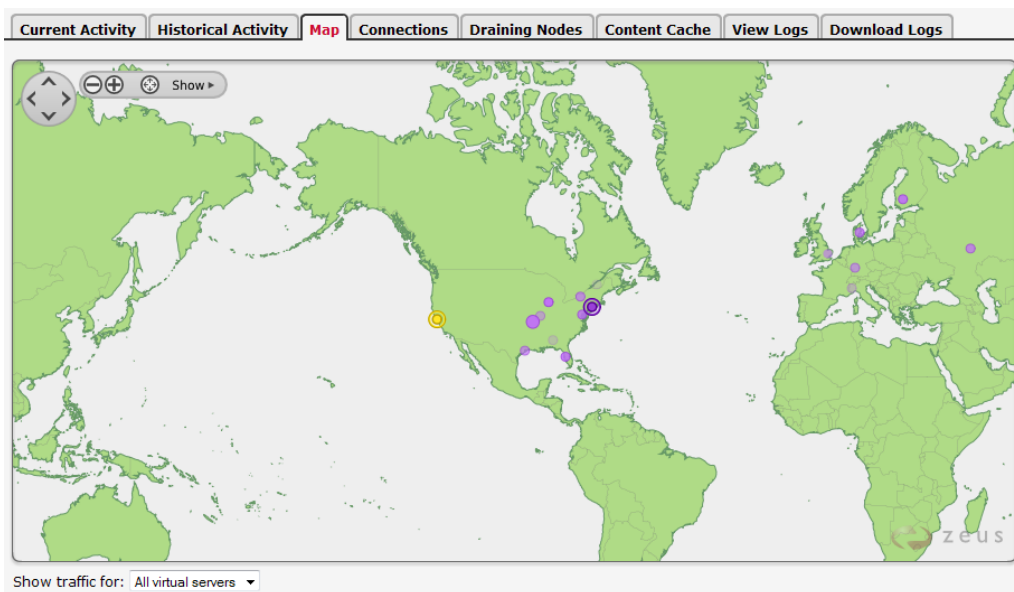
Showing 205 / 205 connections from snapshot

<input checked="" type="checkbox"/>	Time	From	To	State	Request
<input checked="" type="checkbox"/>	254.134:57117	192.168.255.101:80	192.168.254.150	Complete	192.168.254.150/media/emacs-mini.png
<input type="checkbox"/>	254.134:57117	192.168.255.101:80	192.168.254.150	Complete	192.168.254.150/media/mandelbrot2.gif
<input type="checkbox"/>	Process ID	1:14066	192.168.255.101:80	Closed	knowledgehub/xmlsrv/rss2.php
<input checked="" type="checkbox"/>	From	254.134:57116	192.168.255.101:80	Closed	192.168.254.150/index.php/articles?blog=8&page=1&disp=posts&paged=4
<input type="checkbox"/>	Via	254.134:57115	192.168.255.101:80	Closed	192.168.254.150/index.php/articles?blog=8&page=1&disp=posts&paged=4
<input checked="" type="checkbox"/>	To	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/monitor.jpg
<input checked="" type="checkbox"/>	State	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/k_and_r_and_mysql.jpg
<input type="checkbox"/>	Virtual Server	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/ga_intro_small.jpg
<input type="checkbox"/>	Rule	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/5amclock.png
<input type="checkbox"/>	Pool	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/appacceleration.gif
<input type="checkbox"/>	SLM	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/wms.png
<input type="checkbox"/>	Request Bandwidth Class	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/16266104271.jpg
<input type="checkbox"/>	Response Bandwidth Class	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/damianvmworld.png
<input type="checkbox"/>	Bytes In	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/spider.jpg
<input type="checkbox"/>	Bytes Out	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/visio-vss.png
<input type="checkbox"/>	Duration	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/Zeus_ZXTM_shapes.zip
<input type="checkbox"/>	Client Idle Time	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/Stopwatch.jpg
<input type="checkbox"/>	Server Idle Time	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/baldy_fat_geek2.png
<input type="checkbox"/>	Client Keep-alive Number	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/404-fade.png
<input type="checkbox"/>	Server Keep-alive	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/app_error.png
<input type="checkbox"/>	Retries	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/ZXTM_GLB_small.png
<input type="checkbox"/>	Protocol	254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/techworldmap.png
<input type="checkbox"/>		254.134:57115	192.168.255.101:80	Complete	192.168.254.150/media/theft-small.tif

Select and filter on a range of criteria

Display the details of recently processed and in-flight connections

You can display connections on a global map, based on their source address (this new capability is core to Zeus Traffic Manager, and not just restricted to the Real Time Analytics option).



Plot traffic sources, per-service, in real time

The real-time analytics maintains a much larger list of recent connections, and you can optionally decide to record much more detail about each connection – the precise request and response data, and detailed timing information describing how the connection was processed and how long each stage took.

Connection Summary
Unfold All / Fold All

This section shows a summary of a particular connection.

Time: 07-Sep 04:46:31	Traffic Manager: zeus1-ny	Process ID: 2666
Protocol: HTTP	State: Complete	
From: 194.233.245.133:52530	Via: 192.168.254.150:80	To: 192.168.255.101:80
Virtual Server: KnowledgeHub	Rule: None	Pool: KnowledgeHub New York
SLM: None	Response Bandwidth Class: None	Request Bandwidth Class: None
Duration: 0 ms	Client Idle Time: 0 s	Server Idle Time: None
Client Keep-alive Number: 15	Server Keep-alive: Yes	Retries: 0
Bytes In: 243	Bytes Out: 214	
Response Code: 200		
Request: 192.168.254.150/media/16266104271.jpg		

Request tracing

This table shows a trace of significant connection processing events.

Time	Event	Description
0.006 ms	HTTPReqGot	HTTP headers read from client
0.013 ms	HTTPReqParsed	HTTP headers parsed
0.023 ms	RuleRun (Slashdot Throttle)	Rule started
0.036 ms	RuleRun (Fake Source Address)	Rule started
0.113 ms	RuleRun (Redirect requests)	Rule started
0.158 ms	RulesFinished	All rules finished
0.167 ms	HTTPReqBuilt	HTTP request constructed for back-end server
0.179 ms	NodeReuseConn (webserver1:80)	Connection to back-end node re-used
0.508 ms	HTTPResponseGot	HTTP response headers read from back-end server
0.520 ms	RuleRun (Cloak SSN)	Rule started
0.528 ms	RuleRun (Insert RSS feed)	Rule started
0.534 ms	RulesFinished	All rules finished
0.552 ms	End	Request finished

Request Details

View key statistics about a transaction, internal timing details, and full request and response headers

All of the filtering criteria you apply to connections can be implemented in TrafficScript. Once you have identified problem traffic, it is straightforward to then write a TrafficScript rule that filters traffic in an identical fashion, so that you can then drop it, apply a rate limit or rewrite it as required.

Application Auto-Scaling Module

Elastic virtual or cloud platforms offer the potential to scale services up and down in response to demand. This scalability promises to reduce the costs of hosting the service, and an assurance that large spikes in demand can be accommodated without manual intervention.

Zeus' new Application Auto-Scaling module will monitor the performance of a service running on a supported virtual or cloud platform. When the performance falls outside the desired service level, Zeus Traffic Manager can then initiate an auto-scaling action, requesting that the platform deploys additional instances of the service; Zeus Traffic Manager will automatically load balance traffic to the new instances as soon as they are available.

The auto scaling can also scale a service down when it is over-provisioned by removing under-utilized instances. All scaling actions are governed by user-defined limits on the minimum and maximum numbers of servers, and user-defined 'settle times' so that a service is not scaled too quickly in response to a short burst in traffic.

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.

autoscale!enabled: Yes No

Whether or not auto-scaling is being handled by an external system. Set this value to Yes if all aspects of auto-scaling are handled by an external system, such as RightScale. If set to No, the traffic manager will determine when to scale the pool and will communicate with the cloud provider to create and destroy nodes as necessary.

autoscale!external: Yes No

Select the Cloud API and account you want to create nodes with.

Cloud Credentials: - None Selected -

[Manage Cloud Credentials](#)

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.

autoscale!ipstouse: Public IP addresses

The port number to use for each node in this auto-scaled pool.

autoscale!port:

The minimum number of nodes in this auto-scaled pool

autoscale!min_nodes:

The maximum number of nodes in this auto-scaled pool

autoscale!max_nodes:

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.

autoscale!response_time: milliseconds

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.

autoscale!scaledown_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.

autoscale!scaleup_level: %

The time period in seconds after the instigation of a re-size during which no further changes will be made to the pool size.

autoscale!refractory: seconds

The time period in seconds for which a change condition must persist before the change is actually instigated.

autoscale!hysteresis: seconds

The Auto-Scaling settings define when a service should be scaled up or down

Use case – social networking application running on Amazon EC2

A developer deploys a social networking application on Amazon's EC2 cloud, hoping that viral marketing programs will attract a large number of users. Zeus' Application Auto-Scaling allows him to run the application initially on a small number of Amazon instances.

Several weeks later, the viral marketing begins to succeed and the number of users increases rapidly. Without any manual intervention, Zeus begins to deploy more instances of the application to maintain the required level of service, scaling from 4 instances to 60.

Traffic to the application varies significantly with the time of day. Throughout each day, Zeus monitors performance and ensures that the application is never over-provisioned, saving several hundred dollars each day in hosting costs.

Use case – cloud bursting to Rackspace

An enterprise runs a public web service from a corporate datacenter, want to be able to burst traffic to a public cloud when resources are constrained or if traffic grows dramatically.

They run an instance of Zeus Traffic Manager in the RackSpace cloud, with a single instance of their web service which they test periodically to ensure it is operating correctly. When they wish to burst traffic to RackSpace, they switch DNS records (or use Zeus Multi-Site Manager to do so for them) to direct traffic away from their corporate datacenter.

Zeus' autoscaling continually monitors the SLA delivered by the RackSpace instance. If the SLA drops because of the volume of traffic, Zeus automatically provisions additional instances of the application so that the SLA is maintained.

When traffic falls off from the RackSpace instance, Zeus decommissions the unneeded instances, but ensures that at least one instance is running at any time.

Zeus users – What you need to know

Zeus Traffic Manager monitors the response time from server nodes, in a similar fashion to the Service Level Monitoring feature. Auto-scaling is configured on a per-pool basis; each pool monitors the service level of its nodes and then initiates the scaling action based on user-defined parameters (desired response time, thresholds, wait times, maximum and minimum server limits).

When the traffic manager cluster determines that a pool should be scaled, it calls a driver script using Zeus' auto-scaling API. Zeus Traffic Manager version 7 includes packaged drivers for Amazon EC2 and Rackspace, and also integrates with the scaling capabilities of RightScale's management platform.

A user can plug into Zeus' auto-scaling capability to integrate with other virtual or cloud platforms, and Zeus intends to add built-in support for additional platforms in future releases.

TrafficScript Data Structures and Code Libraries

TrafficScript is the high-performance, embedded programming language in Zeus Traffic Manager that is used to create sophisticated traffic management rules. Zeus chose to implement a language runtime from scratch rather than using a third-party language such as TCL or Perl because this offered big performance improvements, tight integration with the traffic manager and more efficient operations.

By design, the TrafficScript language has a similar syntax to common lightweight languages such as Perl, JavaScript and PHP.

The Zeus 7 release adds built-in support for arrays and hashes (also known as 'associative arrays' or 'maps'), and the ability to place TrafficScript code in external libraries and use it within one or more rules.

Use case 1 – managing a list of redirects

Declare a hash table inline

Look up values using square brackets []

```

Rule:
1 $urlmap = [
2   "/zeus7" => "/products/traffic-manager/",
3   "/msm"   => "/products/multi-site-manager/",
4   "/knowledgehub" => "http://knowledgehub.zeus.com/",
5   "/zead"  => "/products/zeus-elastic-application-delivery.html"
6 ];
7
8 $spath = http.getPath();
9
10 $newurl = $urlmap[ $spath ];
11 if( $newurl ) http.redirect( $newurl );
12
13

```

Use case 2 – creating a library to manage an external list

The following rule declares a function `get()` (and an internal function `readFile()`) in the library named `ResourceTable`:

Declare a function...

This function returns the contents of the file from a local cache

Helper function to read and parse the file

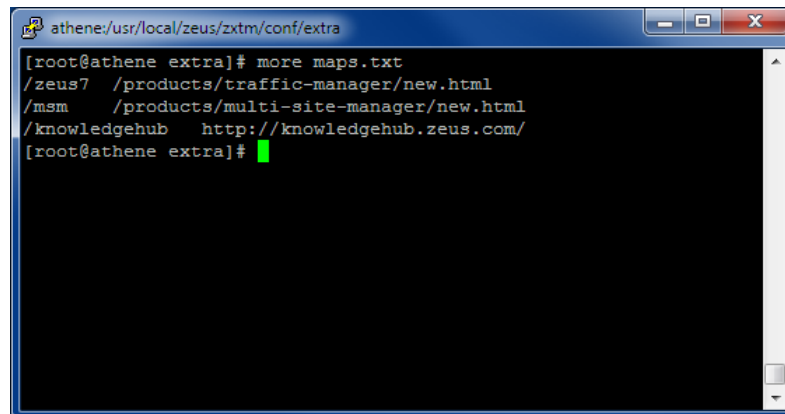
```

Rule:
1 sub get( $filename ) {
2   $key = "resourcetable:". $filename.sys.getPid(); # generate unique key
3
4   if( resource.getMd5( $filename ) != data.get( $key."md5" ) ) {
5     #data has changed
6     $h = readFile( $filename );
7
8     data.set( $key, $h );
9     data.set( $key."md5", resource.getMd5( $filename ) );
10    return $h;
11  }
12
13  return data.get( $key );
14 }
15
16 # Return a hash table of the file contents (key value pairs)
17 sub readFile( $filename ) {
18   $h = [];
19
20   foreach( $line in resource.getLines( $filename ) ) {
21     if( string.regexmatch( $line, '^[^\s]*\s+(.*)$' ) ) $h[$1] = $2;
22   }
23   return $h;
24 }
25

```

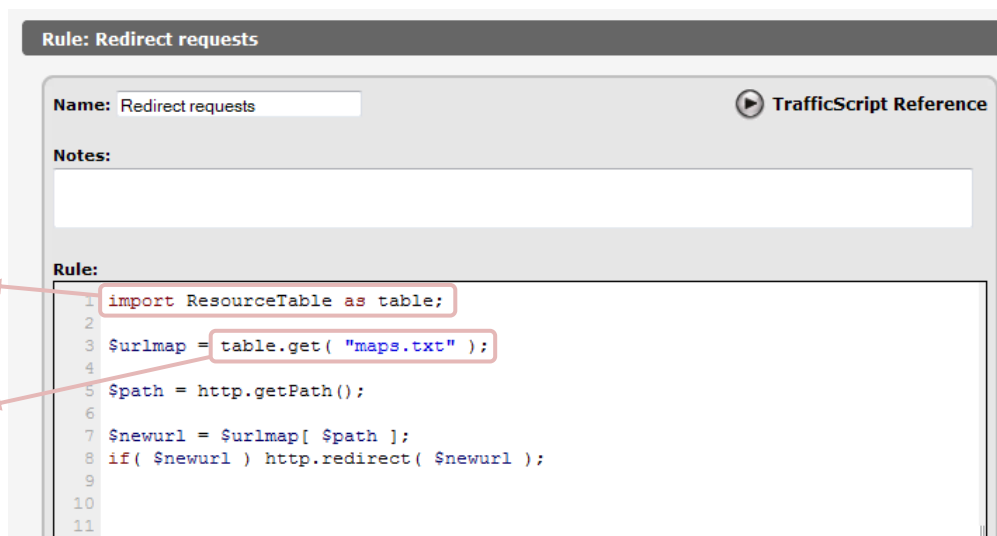
The `readFile()` function reads a table from disk and returns the contents as a TrafficScript hash. The `get()` function wraps a caching layer around the `readFile()` function so that the table is cached internally and only re-read when the disk file changes.

The `maps.txt` file in the `conf/extra` resource directory contains a list of URL mappings:



```
athene:/usr/local/zeus/zxtm/conf/extra
[root@athene extra]# more maps.txt
/zeus7 /products/traffic-manager/new.html
/msm /products/multi-site-manager/new.html
/knowledgehub http://knowledgehub.zeus.com/
[root@athene extra]#
```

Finally, the following TrafficScript rule uses the ResourceTable library above to efficiently look entries up in the `maps.txt` resource file:



```
Rule: Redirect requests
Name: Redirect requests
Notes:
Rule:
1 import ResourceTable as table;
2
3 $urlmap = table.get( "maps.txt" );
4
5 $path = http.getPath();
6
7 $newurl = $urlmap[ $path ];
8 if( $newurl ) http.redirect( $newurl );
9
10
11
```

Use functions from the ResourceTable library
Call the `get()` function from ResourceTable

Zeus users – what you need to know

The addition of arrays, hashes and libraries does not change any of the existing TrafficScript syntax. Your existing rules will continue to work as before, unmodified.

- Any situations where you find yourself duplicating code in several rules are good candidates for extracting the code out into a TrafficScript library.
- Any situations where you find yourself iterating through large amounts of data, overusing the `'data.set()'` and `'data.get()'` functions or performing complex processing are ideal candidates for simplification using array or hash datastructures.

An aside – why create your own language?

Why did Zeus decide to create their own language (TrafficScript), rather than reusing one of the many third party languages available – TCL, Perl, JavaScript, etc?

There are a number of reasons why we decided to do this:

1. **Performance:** TrafficScript is a very lightweight, minimal language with a simple type system, no objects and no unnecessary features. TrafficScript rules are compiled down to a very simple linear byte code that can be executed rapidly with very few run-time checks. The TrafficScript execution environment can be created and destroyed very quickly; much faster than the runtime for a complex third party language like TCL or Perl, and it has to be created and destroyed every time a rule is run.
2. **Integration:** Again to improve performance, creating our own language gave us much more control over things like memory copies, calling internal Zeus functions and garbage collection.

TrafficScript variables are backed directly onto internal datastructures in the traffic management kernel – there's no overhead in creating variables and copying data into the memory space of a third party runtime. Similarly, type consistency means that TrafficScript functions can very quickly call internal functions in the Zeus kernel; there is no need for an intermediate translation layer. Finally, garbage collection is extremely efficient; variables are backed by memory pools with appropriate lifetimes; for example, all of the connection-specific data is stored in a connection memory pool and disposed of in one operation when the connection completes. No need for a complex garbage collector or complex scoping rules.

3. **Pre-emption/cooperative multitasking:** Perhaps the least understood, but certainly the most important ability of the TrafficScript language. Like many high-performance network applications, Zeus Traffic Manager is essentially a single-threaded, single process application that multiplexes thousands of connections simultaneously (strictly speaking, Zeus scales to run one process per core, making light use of shared memory, but the explanation still applies).

The problem with single threaded architectures like Zeus' is that any operation that blocks (for example, waiting for network data) will stall the entire application. Nevertheless, we wanted to provide you with TrafficScript functions that would block, such as `http.getResponse()` or `tcp.write()`, rather than obliging you to write rules in an event-driven fashion.

The TrafficScript runtime implements co-operative multitasking, so that when you call a blocking TrafficScript function your rule is silently stopped. The Traffic Manager proceeds to manage other connections until the event you were waiting for completes (such as reading an HTTP response). Then your rule is restarted precisely where it left off. This means that you can write simple, intuitive rules, and TrafficScript abstracts and hides the complexities of managing transactions in a non-blocking fashion.

For more information: [Zeus KnowledgeHub: Why create a new language?](#)

Connection Concurrency Limiting

Many applications, such as the Apache webserver and many application servers, have a built-in concurrency limit. They simply cannot manage more than a certain number of concurrent connections at the same time; when this number is passed, end-user performance degrades rapidly and the server can be overwhelmed and take a long time to recover.

Zeus connection acceleration capabilities, coupled with the efficient multiplexing design mentioned above, does a good job of reducing a large number of slow, concurrent client-side connections to a very small number of very fast server side connections, so your application is much less susceptible to this type of overload. Nevertheless, a sufficiently large spike of traffic can still render a fragile application unavailable.

Zeus' new Connection Concurrency Limiting capability lets you specify an absolute maximum number of concurrent connections for each node. If the number of active transactions exceeds this number, surplus transactions will be queued internally by the traffic manager until resources become available on the 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.

max_connections_pernode:

The maximum time to keep a connection queued in seconds.

queue_timeout: seconds

The maximum number of connections that can be queued due to connections limits. A value of 0 means unlimited queue size.

max_queue_size:

Specify how many concurrent connections this traffic manager can make to a node

Zeus users – what you need to know

Connection Concurrency Limiting is configured on a per-pool basis. Connection counts are shared between the various Zeus processes on a server, but are not shared between the Zeus systems in a local cluster. Therefore, if you have two Zeus systems running in Active-Active mode, you should define a connection limit that is half the desired amount; active-passive mode is much preferred for this reason.

If a node appears in two or more pools, connection limits are applied independently for each pool. Therefore, you should avoid adding the same node to multiple pools. Where you currently use multiple pools to apply different settings (IP transparency, session persistence etc) to a node, you may need to merge these into one pool and apply these settings conditionally using a TrafficScript rule.

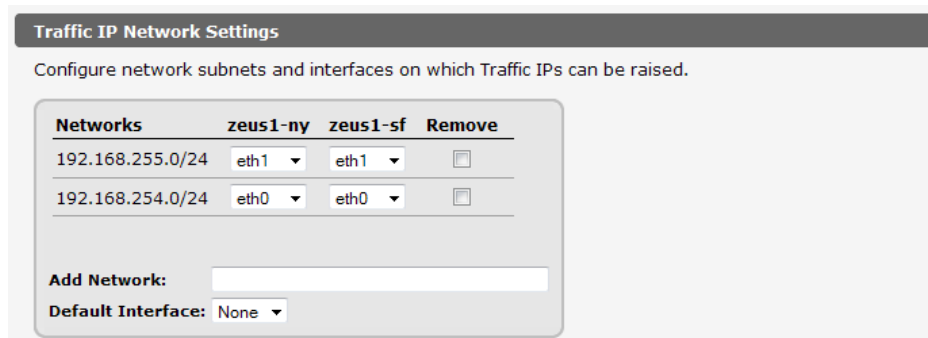
When the limit is exceeded, connections are queued internally, with a maximum backlog and wait time. Connections that exceed the backlog or wait time are discarded.

Reduced IP address requirements for Fault Tolerance

Zeus' TrafficCluster fault tolerance ensures that traffic to your published Traffic IP Addresses is never dropped, even if one or more of the traffic managers fail. Traffic IP addresses are often externally routable IP addresses.

Prior to version 7, Zeus Traffic Manager would check each network interface and determine which subnet it was on based on the IP addresses already raised on it. You had to configure a permanent external IP address on each interface, in addition to the external Traffic IPs. Because external IP addresses are a scarce resource (due in part to IPv4 address depletion), this was sometimes wasteful.

Version 7 adds a new configuration setting for each traffic manager, where you can manually specify which IP subnets are accessible from each interface. This avoids having to configure a permanent external IP address on each traffic manager (although you may want to do so anyway for administration reasons).



Traffic IP Network Settings
Configure network subnets and interfaces on which Traffic IPs can be raised.

Networks	zeus1-ny	zeus1-sf	Remove
192.168.255.0/24	eth1	eth1	<input type="checkbox"/>
192.168.254.0/24	eth0	eth0	<input type="checkbox"/>

Add Network:

Default Interface: None

New configuration settings per-traffic-manager provide hints for traffic IP addresses

When you are running a large cluster of Zeus Traffic Managers, this can significantly reduce the number of external IP addresses you need to reserve for the machines in the cluster.

Zeus users – what you need to know

Your existing Traffic IP groups will continue to function without any modification.

When you create a new Traffic IP group using a new network subnet, you can avoid provisioning additional IP addresses on each traffic manager.

Fine-grained configuration management

Zeus Traffic Manager configuration is stored in a collection of plain-text configuration files that are managed using the Zeus Administration Interface, Zeus' Control API and Zeus' CLI. Users can back up and restore their entire configuration using any of these interfaces.

Zeus version 7 adds a new command line tool that can extract a single configuration item and all of its dependencies. For example, you could extract the configuration for a particular virtual server, along with the pools, rules and other config items that the virtual server uses. This 'partial configuration' can be checked into a source control system, modified offline, and later re-imported back into the original Zeus cluster or a new one.

The command line tool will help you resolve local configuration differences. For example, if you import a virtual server configuration into a new Zeus cluster, the tool will preserve IP addresses and ports in the new cluster. Filters can be added to give more advanced configuration resolution.

This new configuration management capability gives you much more control over your configuration:

- You can archive key parts (or the entirety) of your configuration into a source control system;
- You can modify configuration files offline;
- You can display the differences between a config copy and the running config on a traffic manager;
- You can build a development/test/production hierarchy of traffic manager clusters, and create command-line processes to migrate configuration from one to the other.

Zeus users – what you need to know

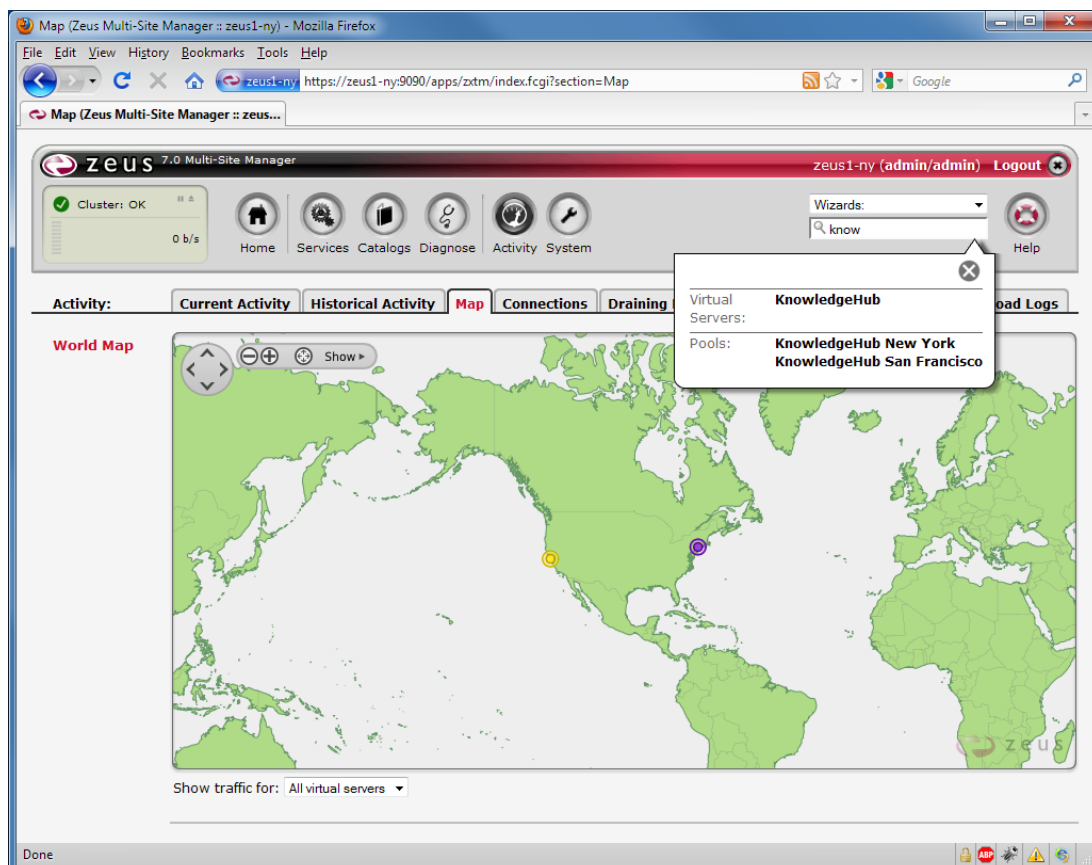
This configuration management is made available using a new command line tool. Zeus configuration files are generally plain text, so can be modified offline if required. Version 7 will include full documentation describing the syntax of configuration files, the keys that are supported and the valid values for each key.

You will be able to manage configuration deployment across multiple locations using tools like Puppet or Chef.

This capability may be used with the Multi-Site Manager to archive configuration. When your traffic managers are grouped into a single cluster, the Multi-site Manager provides a more elegant way to control which configuration is active in each location and visualize configuration and activity across multiple locations.

Other new features

Zeus Traffic Manager 7.0 and the new Multi-Site Manager product are packed with many other new features – UI search, new visualization tools, raw socket I/O functionality in TrafficScript, and much more/



For a full list of the new features in Zeus Traffic Manager 7.0, refer to the release notes and release announcement on the Zeus KnowledgeHub: <http://knowledgehub.zeus.com/>.