Configuring Real Servers and Server Farms

This chapter describes how to configure the servers and server farms and contains these sections:

Configuring Server Farms

A server farm or server pool is a collection of servers that contain the same content. You specify the server farm name when you configure the server farm and add servers to it, and when you bind the server farm to a virtual server. When you configure server farms, do the following:

- Name the server farm.
- Configure a load-balancing algorithm (predictor) and other attributes of the farm.
- Set or specify a set of real servers. (See the "Configuring Real Servers" section on page 5-3.)
- Set or specify the attributes of the real servers.

You also can configure inband health monitoring for each server farm. (See the "Configuring Inband Health Monitoring" section on page 9-8.) You can assign a return code map to a server farm to configure return code parsing. (See the "Configuring HTTP Return Code Checking" section on page 9-9.)

To configure server farms, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)# serverfarm` *`serverfarm-name`*	Creates and names a server farm and enters the server farm configuration mode[1] [2].
Step 2	`Router(config-slb-sfarm)# predictor [roundrobin | leastconns | hash url | hash address [source | destination] [ip-netmask] | forward]]`	Configures the load-balancing prediction algorithm[2]. If not specified, the default is **roundrobin**.

	Command	Purpose
Step 3	`Router(config-slb-sfarm)# nat client client-pool-name`	(Optional) Enables the NAT mode client[2]. (See the "Configuring Client NAT Pools" section on page 5-6.)
Step 4	`Router(config-slb-sfarm)# no nat server`	(Optional) Specifies that the destination IP address is not changed when the load-balancing decision is made.
Step 5	`Router(config-slb-sfarm)# probe probe-name`	(Optional) Associates the server farm to a probe that can be defined by the **probe** command[2].
Step 6	`Router(config-slb-sfarm)# bindid bind-id`	(Optional) Binds a single physical server to multiple server farms and reports a different weight for each one[2]. The **bindid** command is used by DFP.
Step 7	`Router(config-slb-sfarm)# failaction {purge | reassign}`	(Optional) Sets the behavior of connections to real servers that have failed[2].
Step 8	`Router(config-slb-sfarm)# health retries 20 failed 600`	Configures inband health monitoring for all the servers in the server farm.
Step 9	`Cat6k-2(config-slb-sfarm)# retcode-map NAME_OF_MAP`	Configures HTTP return error code checking (requires the configuration of a map of type retcode).
Step 10	`Router(config-slb-sfarm)# real ip_address`	Defines a real server.
Step 11	`Router(config-slb-real)# inservice`	Enables the real servers.
Step 12	`Router# show module csm slot serverfarm serverfarm-name [detail]`	Displays information about one or all server farms.

1. Enter the **exit** command to leave a mode or submode. Enter the **end** command to return to the menu's-top level.
2. The **no** form of this command restores the defaults.

When the least connection predictor is configured, a slow-start mechanism is implemented to avoid sending a high rate of new connections to the servers that have just been put in service. The real server with the fewest number of active connections will get the next connection request for the server farm with the leastconns predictor.

A new environment variable, REAL_SLOW_START_ENABLE is included in this release to control the rate at which a real server ramps up when it put into service. The slow start ramping up is only for a serverfarm configured with the "least-conns" method.

The configurable range for this variable is 0 to 10. The setting of 0 disables the slowstart feature. The value from 1 to 10 specifies how fast the newly activated server should ramp up. The value of 1 is the slowest ramp up rate. The value of 10 specifies that the CSM would assign more requests to the newly activated server. The value of 3 is the default value.

If the configuration value is N, the CSM assigns 2 ^ N (2 raised to the N power) new requests to the newly active server from the start (assuming no connections were terminated at that time). As this server finishes or terminates more connections, a faster ramping occurs. The ramp up stops when the newly activated server has the same number of current opened connections as the other servers in a serverfarm.

This example shows how to configure a server farm, named p1_nat, using the least-connections (**leastconns**) algorithm.

```
Router(config-module-csm)# serverfarm p1_nat
Router(config-slb-sfarm)# predictor leastconns
Router(config-slb-sfarm)# real 10.1.0.105
```

```
Router(config-slb-real)# inservice
Router(config-slb-sfarm)# real 10.1.0.106
Router(config-slb-real)# inservice
```

Configuring Real Servers

Real servers are physical devices assigned to a server farm. Real servers provide the services that are load balanced. When the server receives a client request, it sends the reply to the CSM for forwarding to the client.

You configure the real server in the real server configuration mode by specifying the server IP address and port when you assign it to a server farm. You enter the real server configuration mode from the server farm mode where you are adding the real server.

A real server can be configured as follows:

- no inservvice—The CSM is out of service. There is no sticky and no new connections being applied.

  Note If you specify no inservice, the CSM does not remove open connections. If you want to remove open connections, you must perform that task manually using the **clear module csm** *slot* **connection** command.

- inservice—The CSM is in service. Sticky is allowed and new connections to the module can be made.
- inservice standby—The CSM is in standby. Sticky is allowed. No new connections are allowed.

To configure real servers, perform this task:

	Command	Purpose
Step 1	`Router(config-slb-sfarm)# real` *ip-address* [*port*]	Identifies a real server as a member of the server farm and enters the real server configuration mode. An optional translation port can also be configured[1, 2].
Step 2	`Router(config-slb-real)# weight` *weighting-value*	(Optional) Sets the weighting value for the virtual server predictor algorithm to assign the server's workload capacity relative to the other servers in the server farm if the round robin or least connection is selected[2]. **Note** The only time the sequence of servers starts over at the beginning (with the first server) is when there is a configuration or server state change (either a probe or DFP agent). When the least connection predictor is configured, a slow-start mechanism is implemented to avoid sending a high rate of new connections to the servers that have just been put in service.
Step 3	`Router(config-slb-real)# maxconns` *max-conns*	(Optional) Sets the maximum number of active connections on the real server[2]. When the specified maximum is reached, no more new connections are sent to that real server until the number of active connections drops below the minimum threshold.

	Command	Purpose
Step 4	`Router(config-slb-real)# minconns min-conns`	(Optional) Sets the minimum connection threshold[2].
Step 5	`Router(config-slb-real)# inservice`	Enables the real server for use by the CSM[2] [3].
Step 6	`Router# show module csm slot [sfarm serverfarm-name] [detail]`	(Optional) Displays information about configured real servers. The **sfarm** option limits the display to real servers associated with a particular virtual server. The **detail** option displays detailed real server information.
Step 7	`Router# show module csm slot [vserver virtserver-name] [client ip-address] [detail]`	Displays active connections to the CSM. The **vserver** option limits the display to connections associated with a particular virtual server. The **client** option limits the display to connections for a particular client. The **detail** option displays detailed connection information.

1. Enter the **exit** command to leave a mode or submode. Enter the **end** command to return to the menu's-top level.
2. The **no** form of this command restores the defaults.
3. Repeat Steps 1 through 5 for each real server you are configuring.

This example shows how to create real servers:

```
Router(config-module-csm)# serverfarm serverfarm
Router(config-slb-sfarm)# real 10.8.0.7
Router(config-slb-real)# inservice
Router(config-slb-sfarm)# real 10.8.0.8
Router(config-slb-real)# inservice
Router(config-slb-sfarm)# real 10.8.0.9
Router(config-slb-real)# inservice
Router(config-slb-sfarm)# real 10.8.0.10
Router(config-slb-real)# inservice
Router(config-slb-sfarm)# real 10.1.0.105
Router(config-slb-real)# inservice
Router(config-slb-sfarm)# real 10.1.0.106
Router(config-slb-sfarm)# inservice
Router(config-slb-real)# end
Router# show mod csm slot reals detail
Router# show mod csm slot conns detail
```

The CSM performs graceful server shutdown when a real server is taken out of service using the **no inservice** command. This command stops all new sessions from being load balanced to the real server while allowing existing sessions to complete or time out. New sessions are load balanced to other servers in the server farm for that virtual server.

Note If you specify no inservice, the CSM does not remove open connections. If you want to remove open connections, you must perform that task manually using the **clear module csm** *slot* **conn** command.

The standby state allows the fail action reassignment to reassign connections when a firewall fails. To configure the firewall connection reassignment, you have three options for graceful shutdown:

- Set up a fail action reassignment to a server farm.
- Assign a single real server as a backup for another real server in case of failure.
- The backup real server can be configured with inservice active or in the standby backup state. In standby, this real server would get new connections only when the primary real server failed.

This example shows how to remove a real server from service:

```
Router(config-slb-real)# no inservice
```

For more information on configuring server farms, see the "Configuring Server Farms" section on page 5-1.

The CSM also performs a graceful server shutdown when a real server fails a health probe and is taken out of service. For more information on configuring CSM health probes, see the "Configuring Probes for Health Monitoring" section on page 9-1.

If a client making a request is stuck to an out-of-service server (using a cookie, SSL ID, source IP, etc), this connection is balanced to an in-service server in the farm. If you want to be stuck to an out-of-service server, enter the **inservice standby** command. When you enter the **inservice standby** command, no connections are sent to the standby real server with the exception of those connections that are stuck to that server and those servers with existing connections. After the specified standby time, you can use the **no inservice** command to allow only existing sessions to be sent to that real server. Sticky connections are then sent to an in-service real server in the server farm.

Configuring Dynamic Feedback Protocol

When you configure the Dynamic Feedback Protocol (DFP), the servers can provide feedback to the CSM to enhance load balancing. DFP allows host agents (residing on the physical server) to dynamically report the change in status of the host systems providing a virtual service.

Note A DFP agent may be on any host machine. A DFP agent is independent of the IP addresses and port numbers of the real servers that are managed by the agent. DFP Manager is responsible for establishing the connections with DFP agents and receiving load vectors from DFP agents.

To configure DFP, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)# dfp [password password]`	Configures DFP manager, supplies an optional password, and enters the DFP agent submode[1, 2].
Step 2	`Router(config-slb-dfp)# agent ip-address port [activity-timeout [retry-count [retry-interval]]]`	Configures the time intervals between keepalive messages, the number of consecutive connection attempts or invalid DFP reports, and the interval between connection attempts[2].
Step 3	`Router# show module csm slot dfp [agent [detail | ip-address port] | manager [ip_addr] | detail | weights]`	Displays DFP manager and agent information.

1. Enter the **exit** command to leave a mode or submode. Enter the **end** command to return to the menu's-top level.
2. The **no** form of this command restores the defaults.

This example shows how to configure the dynamic feedback protocol:

```
Router(config-module-csm)# dfp password password
Router(config-slb-dfp)# agent 123.234.34.55 5 6 10 20
Router(config-slb-dfp)# exit
```

Configuring Client NAT Pools

When you configure client Network Address Translation (NAT) pools, NAT converts the source IP address of the client requests into an IP address on the server-side VLAN. Use the NAT pool name in the serverfarm submode of the **nat** command to specify which connections need to be configured for client NAT pools.

To configure client NAT pools, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)# natpool` *pool-name start-ip end-ip* `netmask` *mask*	Configures a content-switching NAT. You must create at least one client address pool to use this command[1, 2].
Step 2	`Router(config-module-csm)# serverfarm` *serverfarm-name*	Enters the serverfarm submode to apply the client NAT.
Step 3	`Router(config-slb-sfarm)# nat client` *clientpool-name*	Associates the configured NAT pool with the server farm.
Step 4	`Router# show module csm natpool [name` *pool-name*`] [detail]`	Displays the NAT configuration.

1. Enter the **exit** command to leave a mode or submode. Enter the **end** command to return to the menu's-top level.
2. The **no** form of this command restores the defaults.

This example shows how to configure client NAT pools:

```
Router(config)# natpool pool1 102.36.445.2 102.36.16.8 netmask 255.255.255.0
Router(config)# serverfarm farm1
Router(config-slb-sfarm)# nat client pool1
```

HTTP header insert is a feature that provides the CSM with the ability to insert information such as the client's IP address into the HTTP header. You configure the HTTP header insert from within the header map. See the "HTTP Header Insert" section on page 8-15 for configuration information.

Configuring Server-Initiated Connections

The NAT for the server allows you to support connections initiated by real servers and to provide a default configuration used for servers initiating connections that do not have matching entries in the server NAT configuration. By default, the CSM allows server-originated connections without NAT.

To configure NAT for the server, perform this task

	Command	Purpose
Step 1	`Router(config)# static [drop | nat [ip-address | virtual]]`	Configures the server-originated connections. Options include dropping the connections, configuring them with NAT with a given IP address, or with the virtual IP address that they are associated with[1, 2].
Step 2	`Router(config-slb-static)# real ip-address [subnet-mask]`	Configures the static NAT submode where the servers will have this NAT option. You cannot use the same real server with multiple NAT configuration options.

1. Enter the **exit** command to leave a mode or submode. Enter the **end** command to return to the menu's top level.
2. The **no** form of this command restores the defaults.

Configuring URL Hashing

When you choose a server farm for a connection, you can select a specific real server in that server farm. You can choose least connections, round robin, or URL hashing to select a real server.

URL hashing is a load-balancing predictor for Layer 7 connections. You can configure URL hashing on the CSM on a server farm-by-server farm basis. The CSM chooses the real server by using a hash value based on a URL. This hash value may be computed on the entire URL or on a portion of it. To select only a portion of the URL for hashing, you can specify the beginning and ending patterns in the URL so that only the portion of the URL from the specified beginning pattern through the specified ending pattern is hashed. The CSM supports URL hashing in software release 2.1(1).

Unless you specify a beginning and an ending pattern (see the "Configuring Beginning and Ending Patterns" section on page 5-8), the entire URL is hashed and used to select a real server.

Configuring a URL Hashing Predictor

You must configure URL hashing for all server farms that will be using the URL hashing predictor, regardless of whether they are using the entire URL or a beginning and ending pattern.

To configure URL hashing as a load-balancing predictor for a server farm, perform this task:

Command	Purpose
`Router(config-slb-sfarm)# predictor hash url`	Configures the URL hashing and load-balancing predictor for a server farm.

This example shows how to configure the URL hashing and load-balancing predictor for a server farm:

```
Router(config)# mod csm 2
Router(config-module-csm)# serverfarm farm1
Router(config-slb-sfarm)# predictor hash url
Router(config-slb-sfarm)# real 10.1.0.105
Router(config-slb-real)# inservice
Router(config-slb-real)# exit
```

Cache servers perform better using URL hashing. However, the hash methods do not recognize weight for the real servers. The weight assigned to the real servers is used in the round-robin and least-connection predictor methods.

Note The only time the sequence of servers starts over at the beginning (with the first server) is when there is a configuration or server state change (either a probe or DFP agent).

To create different weights for real servers, you can list multiple IP addresses of the cache server in the server farm. You can also use the same IP address with a different port number.

Note Server weights are not used for hash predictors.

To configure real servers with a weight when using the URL hash predictor, perform this task:

	Command	Purpose
Step 1	`Router(config-slb-sfarm)#` **`serverfarm MYFARM`**	Creates a server farm named MYFARM.
Step 2	`Router(config-slb-sfarm)#` **`real 1.1.1.1 80`**	Specifies the real server at port 80.
Step 3	`Router(config-slb-sfarm)#` **`inservice`**	Enables the real server in service.
Step 4	`Router(config-slb-sfarm)#` **`real 1.1.1.1 8080`**	Specifies the real server at port 8080.
Step 5	`Router(config-slb-sfarm)#` **`inservice`**	Enables the real server in service.

Configuring Beginning and Ending Patterns

You configure a beginning and ending pattern at the virtual server level. The pattern you define applies to all the server farms assigned to all of the policies in that virtual server that have URL hashing enabled.

The beginning and ending pattern delimits the portion of the URL that will be hashed and used as a predictor to select a real server from a server farm that belongs to any policy assigned to that virtual server.

To hash a substring of the URL instead of the entire URL, specify the beginning and ending patterns in **vserver vserver-name** submode with the **url-hash begin-pattern** *pattern-a* command and **url-hash end-pattern** *pattern-b* command. Hashing occurs at the start of the beginning pattern and goes to the ending pattern.

For example, in the following URL, if the beginning pattern is **c&k=**, and the ending pattern is **&**, only the substring **c&k=c** is hashed:

http://quote.yahoo.com/q?s=csco&d=**c&k=c1**&t=2y&a=v&p=s&l=on&z=m&q=l\

Note Beginning and ending patterns are restricted to fixed constant strings. General regular expressions cannot be specified as patterns. If no beginning pattern is specified, hashing begins at the beginning of the URL. If no ending pattern is specified, hashing ends at the end of the URL.

This example shows how to configure beginning and ending patterns for URL hashing:

```
Router(config-module-csm)#
Router(config-module-csm)# vserver vs1
Router(config-slb-vserver)# virtual 10.1.0.81 tcp 80
Router(config-slb-vserver)# url-hash begin-pattern c&k= end-pattern &
Router(config-slb-vserver)# serverfarm farm1
Router(config-slb-vserver)# inservice
Router(config-slb-vserver)#
Router(config-slb-vserver)# exit
Router(config-module-csm)# exit
```