Keepalived User Guide

Release 1.4.3

Alexandre Cassen and Contributors

August 13, 2022

Contents

CHAPTER 1

Introduction

Load balancing is a method of distributing IP traffic across a cluster of real servers, providing one or more highly available virtual services. When designing load-balanced topologies, it is important to account for the availability of the load balancer itself as well as the real servers behind it.

Keepalived provides frameworks for both load balancing and high availability. The load balancing framework relies on the well-known and widely used Linux Virtual Server (IPVS) kernel module, which provides Layer 4 load balancing. Keepalived implements a set of health checkers to dynamically and adaptively maintain and manage load balanced server pools according to their health. High availability is achieved by the Virtual Redundancy Routing Protocol (VRRP). VRRP is a fundamental brick for router failover. In addition, keepalived implements a set of hooks to the VRRP finite state machine providing low-level and high-speed protocol interactions. Each Keepalived framework can be used independently or together to provide resilient infrastructures.

In this context, load balancer may also be referred to as a *director* or an *LVS router*.

In short, Keepalived provides two main functions:

- Health checking for LVS systems
- Implementation of the VRRPv2 stack to handle load balancer failover

CHAPTER 2

Software Design

Keepalived is written in pure ANSI/ISO C. The software is articulated around a central I/O multiplexer that provides realtime networking design. The main design focus is to provide a homogenous modularity between all elements. This is why a core library was created to remove code duplication. The goal is to produce a safe and secure code, ensuring production robustness and stability.

To ensure robustness and stability, daemon is split into 4 distinct processes:

- A minimalistic parent process in charge with forked children process monitoring.
- Up to three child processes, one responsible for VRRP framework, one for healthchecking and IPVS configuration, and one for BFD.

Each child process has its own scheduling I/O multiplexer, that way VRRP scheduling jitter is optimized since VRRP and BFD scheduling are more sensitive/critical than healthcheckers. This split design minimalizes for healthchecking the usage of foreign libraries and minimalizes its own action down to an idle mainloop in order to avoid malfunctions caused by itself.

The parent process monitoring framework is called watchdog. If the parent process detects that a child has terminated it simply restarts child process:

```
PID         111     Keepalived  <-- Parent process monitoring children
            112     \_ Keepalived   <-- VRRP child
            113     \_ Keepalived   <-- Healthchecking child
            114     \_ Keepalived   <-- BFD child
```

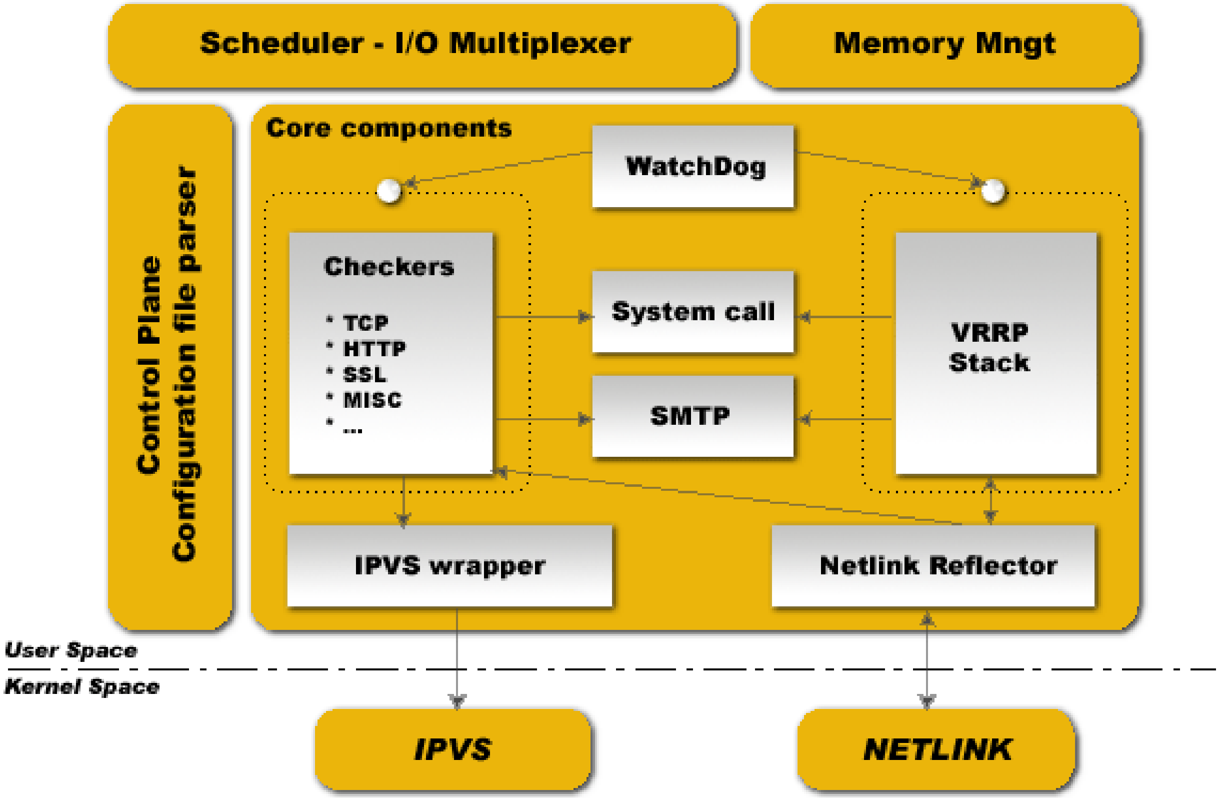
2.1 Kernel Components

Keepalived uses four Linux kernel components:

1. LVS Framework: Uses the getsockopt and setsockopt calls to get and set options on sockets.
2. Netfilter Framework: IPVS code that supports NAT and Masquerading.
3. Netlink Interface: Sets and removes VRRP virtual IPs on network interfaces.

4. Multicast: VRRP advertisements are sent to the reserved VRRP MULTICAST group (224.0.0.18).

2.2 Atomic Elements



2.2.1 Control Plane

Keepalived configuration is done through the file keepalived.conf. A compiler design is used for parsing. Parser work with a keyword tree hierarchy for mapping each configuration keyword with specifics handler. A central multi-level recursive function reads the configuration file and traverses the keyword tree. During parsing, configuration file is translated into an internal memory representation.

2.2.2 Scheduler - I/O Multiplexer

For each process, all the events are scheduled into the same process. Keepalived is network routing software, it is so close to I/O. The design used here is a central epoll_wait(. . .) that is in charge of scheduling all internal tasks. POSIX thread libs are NOT used. This framework provides its own thread abstraction optimized for networking purpose.

2.2.3 Memory Management

This framework provides access to some generic memory management functions like allocation, reallocation, release,. . . This framework can be used in two modes: normal_mode & debug_mode. When using debug_mode it provides a strong way to eradicate and track memory leaks. This low-level env provides buffer under-run protection by tracking allocation and release of memory. All the buffers used are length fixed to prevent against eventual buffer-overflow.

2.2.4 Core Components

This framework defines some common and global libraries that are used in all the code. Those libraries are html parsing, link-list, timer, vector, string formating, buffer dump, networking utils, daemon management, pid handling, low-level TCP layer4. The goal here is to factorize code to the max to limit as much as possible code duplication to increase modularity.

2.2.5 Checkers

This is one of the main Keepalived functionality. Checkers are in charge of adding, removing and changing the weight of realservers. There are several types of checkers, most of which relate to realserver healthchecking. A checker tests if realserver is alive, this test either ends on a binary decision: remove or add realserver from/into the LVS topology, or changing the weight of the realserver. The internal checker design is realtime networking software, it uses a fully multi-threaded FSM design (Finite State Machine). This checker stack provides LVS topology manipulation according to layer4 to layer5/7 test results. It's run in an independent process monitored by the parent process.

2.2.6 VRRP Stack

The other most important Keepalived functionality. VRRP (Virtual Router Redundancy Protocol: RFC2338/RFC3768/RFC5798) is focused on director takeover, it provides low-level design for router backup. It implements full IETF RFC5798 standard with some provisions and extensions for LVS and Firewall design (with legacy support for RFC2338, i.e. authentication). It implements the vrrp_sync_group extension that guarantees persistence routing path after protocol takeover. It implements IPSEC-AH using MD5-96bit crypto provision for securing protocol adverts exchange. For more information on VRRP please read the RFC. Important things: VRRP code can be used without the LVS support, it has been designed for independent use. It's run in an independent process monitored by parent process.

2.2.7 BFD Stack

An implementation of BFD (Bidirectional Forwarding and Detection: RFC5880). This can be used by both the VRRP process as a tracker for VRRP instance(s) and by the checker process as a checker for realserver. It's run in an independent process monitored by parent process.

2.2.8 System Call

This framework offers the ability to launch extra system script. It is mainly used in the MISC checker. In VRRP framework it provides the ability to launch extra script during protocol state transition. The system call is done into a forked process to not pertube the global scheduling timer.

2.2.9 Netlink Reflector

Same as IPVS wrapper. Keepalived works with its own network interface representation. IP address and interface flags are set and monitored through kernel Netlink channel. The Netlink messaging sub-system is used for setting VRRP VIPs. On the other hand, the Netlink kernel messaging broadcast capability is used to reflect into our userspace Keepalived internal data representation any events related to interfaces. So any other userspace (others program) netlink manipulation is reflected our Keepalived data representation via Netlink Kernel broadcast (RTMGRP_LINK & RTMGRP_IPV4_IFADDR).

2.2.10 SMTP

The SMTP protocol is used for administration notification. It implements the IETF RFC821 using a multi-threaded FSM design. Administration notifications are sent for healthcheckers activities and VRRP protocol state transition. SMTP is commonly used and can be interfaced with any other notification sub-system such as GSM-SMS, pagers, etc.

2.2.11 IPVS Wrapper

This framework is used for sending rules to the Kernel IPVS code. It provides translation between Keepalived internal data representation and IPVS rule_user representation. It uses the IPVS libipvs to keep generic integration with IPVS code.

2.2.12 IPVS

The Linux Kernel code provided by Wensong from LinuxVirtualServer.org OpenSource Project. IPVS (IP Virtual Server) implements transport-layer load balancing inside the Linux kernel, also referred to as Layer-4 switching.

2.2.13 NETLINK

The Linux Kernel code provided by Alexey Kuznetov with its very nice advanced routing framework and sub-system capabilities. Netlink is used to transfer information between kernel and user-space processes. It consists of a standard sockets-based interface for userspace processes and an internal kernel API for kernel modules.

2.2.14 Syslog

All keepalived daemon notification messages are logged using the syslog service.

2.3 Healthcheck Framework

Each health check is registered to the global scheduling framework. These health check worker threads implement the following types of health checks:

TCP_CHECK Working at layer4. To ensure this check, we use a TCP Vanilla check using nonblocking/timed-out TCP connections. If the remote server does not reply to this request (timed-out), then the test is wrong and the server is removed from the server pool.

HTTP_GET Working at layer5. Performs a HTTP GET to a specified URL. The HTTP GET result is then summed using the MD5 algorithm. If this sum does not match with the expected value, the test is wrong and the server is removed from the server pool. This module implements a multi-URL get check on the same service. This functionality is useful if you are using a server hosting more than one application servers. This functionality gives you the ability to check if an application server is working properly. The MD5 digests are generated using the genhash utility (included in the keepalived package).

SSL_GET Same as HTTP_GET but uses a SSL connection to the remote webservers.

MISC_CHECK This check allows a user-defined script to be run as the health checker. The result must be 0 or 1. The script is run on the director box and this is an ideal way to test in-house applications. Scripts that can be run without arguments can be called using the full path (i.e. /path_to_script/script.sh). Those requiring arguments need to be enclosed in double quotes (i.e. "/path_to_script/script.sh arg 1 … arg n ")

SMTP_CHECK This check ensures that an SMTP server can be connected to and the initial SMTP handshake completed.

DNS_CHECK This check queries a DNS server for the configured name of the specified type (e.g. A, AAAA, MX record).

BFD_CHECK This is updated by the BFD process, and allows a realserver to be removed if the BFD session goes down.

UDP_CHECK This check sends a UDP packet to the specified remote host/port. It can be configured to require a specific response, or to fail if an ICMP error is returned.

PING_CHECK This check sends and ICMP echo request and will fail if an appropriate ICMP echo response is not received.

FILE_CHECK This check monitors a file using inotify(). If the file is modified or created, its contents are read and interpreted as a numeric value. This can either indicate the realserver should be removed, or its weight changed, depending on the configuration.

The goal for Keepalived is to define a generic framework easily extensible for adding new checkers modules. If you are interested in the development of existing or new checkers, have a look at the *keepalived/check* and *keepalived/trackers* directories in the source:

https://github.com/acassen/keepalived/tree/master/keepalived/check

2.4 Failover (VRRP) Framework

Keepalived implements the VRRP protocol for director failover. Within the implemented VRRP stack, the VRRP Packet dispatcher is responsible for demultiplexing specific I/O for each VRRP instance.

From RFC5798, VRRP is defined as:

```
"VRRP specifies an election protocol that dynamically assigns
responsibility for a virtual router to one of the VRRP routers on a LAN.
The VRRP router controlling the IPv4 or IPv6 address(es) associated with
a virtual router is called the Master, and it forwards packets sent to
these IPv4 or IPv6 addresses.  VRRP Master routers are configured with
virtual IPv4 or IPv6 addresses, and VRRP Backup routers infer the
address family of the virtual addresses being carried based on the
transport protocol.  Within a VRRP router, the virtual routers in
each of the IPv4 and IPv6 address families are a domain unto
themselves and do not overlap.  The election process provides dynamic
failover in the forwarding responsibility should the Master become
unavailable.  For IPv4, the advantage gained from using VRRP is a
higher-availability default path without requiring configuration of
dynamic routing or router discovery protocols on every end-host.  For
IPv6, the advantage gained from using VRRP for IPv6 is a quicker
switchover to Backup routers than can be obtained with standard IPv6
Neighbor Discovery mechanisms." [rfc5798]
```

Note: This framework is LVS independent, so you can use it for LVS director failover, even for other Linux routers needing a Hot-Standby protocol. This framework has been completely integrated in the Keepalived daemon for design

& robustness reasons.

The main functionalities provided by this framework are:

- Failover: The native VRRP protocol purpose, based on a roaming set of VRRP VIPs.
- VRRP Instance synchronization: We can specify a state monitoring between 2 or more VRRP Instances, also known as a *VRRP sync group*. It guarantees that the VRRP Instances remain in the same state. The synchronized instances monitor each other.
- Nice Fallback
- Advert Packet integrity: Using IPSEC-AH ICV.
- System call: During a VRRP state transition, an external script/program may be called.

2.4.1 Note on Using VRRP with Virtual MAC Address

To reduce takeover impact, some networking environment would require using VRRP with VMAC address. To reach that goal Keepalived VRRP framework implements VMAC support by the invocation of 'use_vmac' keyword in configuration file.

Internally, Keepalived code will bring up virtual interfaces, each interface dedicated to a specific virtual_router. Keepalived uses Linux kernel macvlan driver to defines these interfaces. It is then mandatory to use kernel compiled with macvlan support.

By default MACVLAN interface are in VEPA mode which filters out received packets whose MAC source address matches that of the MACVLAN interface. Setting MACVLAN interface in private mode will not filter based on source MAC address.

Alternatively, you can specify 'vmac_xmit_base' which will cause the VRRP messages to be transmitted and received on the underlying interface whilst ARP will happen from the VMAC interface.

You may also need to tweak your physical interfaces to play around with well known ARP issues. Keepalived sets the following configuration when using VMACs:

1) Global configuration:

```
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 1
net.ipv4.conf.all.arp_filter = 0
```

2) Physical interface configuration

For the physical ethernet interface running VRRP instance use:

```
net.ipv4.conf.eth0.arp_filter = 1
```

3) VMAC interface

consider the following VRRP configuration:

```
vrrp_instance instance1 {
    state BACKUP
    interface eth0
    virtual_router_id 250
    use_vmac
        vmac_xmit_base          # Transmit VRRP adverts over physical interface
    priority 150
```

(continues on next page)

(continued from previous page)

```
    advert_int 1
    virtual_ipaddress {
        10.0.0.254
    }
}
```

The `use_vmac` keyword will drive keepalived code to create a macvlan interface named *vrrp.250* (default internal paradigm is vrrp.{virtual_router_id}, you can override this naming by giving an argument to 'use_vmac' keyword, eg: use_vmac vrrp250).

CHAPTER 3

Load Balancing Techniques

3.1 Virtual Server via NAT

NAT Routing is used when the Load-Balancer (or LVS Router) has two Network Interface Cards (NICs), one assigned an outside-facing IP address and the other, a private, inside-facing IP address. In this method, the Load-Balancer receives requests from users on the public network and uses network address translation (NAT) to forward those requests to the real servers located on the private network. The replies are also translated in the reverse direction, when the real servers reply to the users' requests.

As a result, an advantage is that the real servers are protected from the public network as they are hidden behind the Load-Balancer. Another advantage is IP address preservation, as the private network can use private address ranges.

The main disadvantage is that the Load-Balancer becomes a bottleneck. It has to serve not only requests but also replies to and from the public users, while also forwarding to and from the private real servers.

3.2 Virtual Server via Tunneling

In Tunneling mode, the Load-Balancer sends requests to real servers through IP tunnel in the former, and the Load-Balancer sends request to real servers via network address translation in the latter.

The main advantage of this method is scalability, Load-Balancer will forward incoming request to farm nodes, latter nodes will then respond directly to the client requests without having to proxy through Load-Balancer. It offers you a way to locate nodes in different networking segments.

The main disadvantage is the cost you will put into it to finally get a working env since it is deeply dependent upon your network architecture.

3.3 Virtual Server via Direct Routing

In Direct Routing, users issue requests to the VIP on the Load-Balancer. The Load-Balancer uses its predefined scheduling (distribution) algorithm and forwards the requests to the appropriate real server. Unlike using NAT Routing, the real servers respond directly to the public users, bypassing the need to route through the Load-Balancer.

The main advantage with this routing method is scalability, as the Load-Balancer does not have the additional responsibility of routing outgoing packets from the real servers to the public users.

The disadvantage to this routing method lies in its ARP limitation. In order for the real servers to directly respond to the public users' requests, each real server must use the VIP as its source address when sending replies. As a result, the VIP and MAC address combination are shared amongst the Load-Balancer itself as well as each of the real servers that can lead to situations where the real servers receive the requests directly, bypassing the Load-Balancer on incoming requests. There are methods available to solve this problem at the expense of added configuration complexity and manageability.

CHAPTER 4

Installing Keepalived

Install keepalived from the distribution's repositories or, alternatively, compile from source. Although installing from the repositories is generally the fastest way to get keepalived running on a system, the version of keepalived available in the repositories are typically a few releases behind the latest available stable version.

4.1 Installing from the Repositories

4.1.1 Installing on Red Hat Enterprise Linux

As of Red Hat 6.4, Red Hat and the clones have included the keepalived package in the base repository. Therefore, run the following to install the keepalived package and all the required dependencies using YUM:

```
yum install keepalived
```

4.1.2 Installing on Debian

Run the following to install the keepalived package and all the required dependencies using Debian's APT package handling utility:

```
apt-get install keepalived
```

4.2 Compiling and Building from Source

In order to run the latest stable version, compile keepalived from source. Compiling keepalived requires a compiler, OpenSSL and the Netlink Library. You may optionally install Net-SNMP, which is required for SNMP support.

4.2.1 Install Prerequisites on RHEL/CentOS/Fedora

On RHEL, Centos, Fedora etc install the following prerequisites (on older systems replace dnf with yum):

```
dnf install curl gcc autoconf automake openssl-devel libnl3-devel \
    iptables-devel ipset-devel net-snmp-devel libnfnetlink-devel file-devel
```

For DBUS support:

```
dnf install glib2-devel
```

For JSON support:

```
dnf install json-c-devel
```

4.2.2 Install Prerequisites on Debian/Ubuntu

On Debian/Ubuntu, install the following prerequisites:

```
apt-get install pkg-config curl gcc autoconf automake libssl-dev libnl-3-dev \
    libnl-genl-3-dev libsnmp-dev libnl-route-3-dev libnfnetlink-dev libipset-dev \
    iptables-dev libsnmp-dev
```

For DBUS support:

```
dnf install libglib2.0-dev
```

4.2.3 Install Prerequisites on Alpine Linux

On Alpine Linux install the following prerequisites:

```
autoconf automake iptables-dev ipset-dev libnfnetlink-dev libnl3-dev musl-dev and
    openssl-dev or libressl-dev
```

For SNMP support:

```
net-snmp-dev (requires libressl-dev and not openssl-dev)
```

4.2.4 Install Prerequisites on Archlinux

On Archlinux run the following to install the required libraries:

```
pacman -S ipset libnfnetlink libnl1
```

For SNMP support:

```
pacman -S net-snmp
```

4.2.5 Build and Install

Use *curl* or any other transfer tool such as *wget* to download keepalived. The software is available at http://www.keepalived.org/download.html or https://github.com/acassen/keepalived. Then, compile the package:

```
curl --progress http://keepalived.org/software/keepalived-1.2.15.tar.gz | tar xz
cd keepalived-1.2.15
./build_setup
./configure
make
sudo make install
```

It is a general recommendation when compiling from source to specify a PREFIX. For example:

```
./configure --prefix=/usr/local/keepalived-1.2.15
```

This makes it easy to uninstall a compiled version of keepalived simply by deleting the parent directory. Additionally, this method of installation allows for multiple versions of Keepalived installed without overwriting each other. Use a symlink to point to the desired version. For example, your directory layout could look like this:

```
[root@lvs1 ~]# cd /usr/local
[root@lvs1 local]# ls -l
total 12
lrwxrwxrwx. 1 root root   17 Feb 24 20:23 keepalived -> keepalived-1.2.15
drwxr-xr-x. 2 root root 4096 Feb 24 20:22 keepalived-1.2.13
drwxr-xr-x. 2 root root 4096 Feb 24 20:22 keepalived-1.2.14
drwxr-xr-x. 2 root root 4096 Feb 24 20:22 keepalived-1.2.15
```

4.2.6 Setup Init Scripts

After compiling, create an init script in order to control the keepalived daemon.

On RHEL:

```
ln -s /etc/rc.d/init.d/keepalived.init /etc/rc.d/rc3.d/S99keepalived
```

On Debian:

```
ln -s /etc/init.d/keepalived.init /etc/rc2.d/S99keepalived
```

Note: The link should be added in your default run level directory.

CHAPTER 5

Keepalived configuration synopsis

The Keepalived configuration file uses the following synopsis (configuration keywords are Bold/Italic):

5.1 Global Definitions Synopsis

```
global_defs {
    notification_email {
        email
        email
    }
    notification_email_from email
    smtp_server host
    smtp_connect_timeout num
    router_id string
}
```

Keyword	Definition	Type
global_defs	identify the global def configuration block	
notification_email	email accounts that will receive the notification mail	List
notification_email_from	email to use when processing “MAIL FROM:” SMTP command	List
smtp_server remote SMTP	server to use for sending mail notifications	alphanum
smtp_connect_timeout	specify a timeout for SMTP stream processing	numerical
router_id	specify the name of the LVS director	string

Email type: Is a string using charset as specified into the SMTP RFC eg: “user@domain.com”

5.2 Virtual Server Definitions Synopsis

```
virtual_server (@IP PORT)|(fwmark num) {
    delay_loop num
    lb_algo rr|wrr|lc|wlc|sh|dh|lblc
    lb_kind NAT|DR|TUN
    (nat_mask @IP)
    persistence_timeout num
    persistence_granularity @IP
    virtualhost string
    protocol TCP|UDP

    sorry_server @IP PORT
    real_server @IP PORT {
        weight num
        TCP_CHECK {
            connect_port num
            connect_timeout num
        }
    }
    real_server @IP PORT {
        weight num
        MISC_CHECK {
            misc_path /path_to_script/script.sh
            (or misc_path “ /path_to_script/script.sh <arg_list>”)
        }
    }
}
real_server @IP PORT {
    weight num
    HTTP_GET|SSL_GET {
        url { # You can add multiple url block
            path alphanum
            digest alphanum
        }
        connect_port num
        connect_timeout num
        retry num
        delay_before_retry num
    }
}
```

Keyword	Definition	Type
virtual_server	identify a virtual server definition block	
fwmark	specify that virtual server is a FWMARK	
delay_loop	specify in seconds the interval between checks	numerical
lb_algo	select a specific scheduler (rr|wrr|lc|wlc…)	string
lb_kind	select a specific forwarding method (NAT|DR|TUN)	string
persistence_timeout	specify a timeout value for persistent connections	numerical
persistence_granularity	specify a granularity mask for persistent connections	
virtualhost	specify a HTTP virtualhost to use for HTTP|SSL_GET	alphanum
protocol	specify the protocol kind (TCP|UDP)	numerical
sorry_server	server to be added to the pool if all real servers are down	
real_server	specify a real server member	
weight	specify the real server weight for load balancing decisions	numerical
TCP_CHECK	check real server availability using TCP connect	
MISC_CHECK	check real server availability using user defined script	
misc_path	identify the script to run with full path	path
HTTP_GET	check real server availability using HTTP GET request	
SSL_GET	check real server availability using SSL GET request	
url	identify a url definition block	
path	specify the url path	alphanum
digest	specify the digest for a specific url path	alphanum
connect_port	connect remote server on specified TCP port	numerical
connect_timeout	connect remote server using timeout	numerical
retry	maximum number of retries	numerical
delay_before_retry	delay between two successive retries	numerical

Note: The “nat_mask” keyword is obsolete if you are not using LVS with Linux kernel 2.2 series. This flag give you the ability to define the reverse NAT granularity.

Note: Currently, Healthcheck framework, only implements TCP protocol for service monitoring.

Note: Type “path” refers to the full path of the script being called. Note that for scripts requiring arguments the path and arguments must be enclosed in double quotes (“).

5.3 VRRP Instance Definitions Synopsis

```
vrrp_sync_group string {
    group {
        string
        string
    }
    notify_master /path_to_script/script_master.sh
        (or notify_master “ /path_to_script/script_master.sh <arg_list>”)
    notify_backup /path_to_script/script_backup.sh
        (or notify_backup “/path_to_script/script_backup.sh <arg_list>”)
    notify_fault /path_to_script/script_fault.sh
```

```
        (or notify_fault “ /path_to_script/script_fault.sh <arg_list>”)
}
vrrp_instance string {
    state MASTER|BACKUP
    interface string
    mcast_src_ip @IP
    lvs_sync_daemon_interface string
    virtual_router_id num
    priority num
    advert_int num
    smtp_alert
    authentication {
        auth_type PASS|AH
        auth_pass string
    }
    virtual_ipaddress { # Block limited to 20 IP addresses
        @IP
        @IP
        @IP
    }
    virtual_ipaddress_excluded { # Unlimited IP addresses
        @IP
        @IP
        @IP
    }
    notify_master /path_to_script/script_master.sh
        (or notify_master “ /path_to_script/script_master.sh <arg_list>”)
    notify_backup /path_to_script/script_backup.sh
        (or notify_backup “ /path_to_script/script_backup.sh <arg_list>”)
    notify_fault /path_to_script/script_fault.sh
        (or notify_fault “ /path_to_script/script_fault.sh <arg_list>”)
}
```

Keyword	Definition	Type
vrrp_instance	identify a VRRP instance definition block	
state	specify the instance state in standard use	
Interface	specify the network interface for the instance to run on	string
mcast_src_ip	specify the src IP address value for VRRP adverts IP header	
lvs_sync_daemon_inteface	specify the network interface for the LVS sync_daemon to run on	string
virtual_router_id	specify to which VRRP router id the instance belongs	numerical
priority	specify the instance priority in the VRRP router	numerical
advert_int	specify the advertisement interval in seconds (set to 1)	numerical
smtp_alert	Activate the SMTP notification for MASTER state transition	
authentication	identify a VRRP authentication definition block	
auth_type	specify which kind of authentication to use (PASS|AH)	
auth_pass	specify the password string to use	string
virtual_ipaddress	identify a VRRP VIP definition block	
virtual_ipaddress_excluded	identify a VRRP VIP excluded definition block (not protocol VIPs)	
notify_master	specify a shell script to be executed during transition to master state	path
notify_backup	specify a shell script to be executed during transition to backup state	path
notify_fault	specify a shell script to be executed during transition to fault state	path
vrrp_sync_group	Identify the VRRP synchronization instances group	string

Path type: A system path to a script eg: “/usr/local/bin/transit.sh <arg_list>”

CHAPTER 6

Keepalived programs synopsis

Keepalived package comes with 2 programs.

6.1 keepalived daemon

The keepalived command line arguments are:

-f, –use-file=FILE Use the specified configuration file. The default configuration file is "/etc/keepalived/keepalived.conf".

-P, –vrrp Only run the VRRP subsystem. This is useful for configurations that do not use IPVS load balancer.

-C, –check Only run the healthcheck subsystem. This is useful for configurations that use the IPVS load balancer with a single director with no failover.

-l, –log-console Log messages to the local console. The default behavior is to log messages to syslog.

-D, –log-detail Detailed log messages.

-S, –log-facility=[0-7] Set syslog facility to LOG_LOCAL[0-7]. The default syslog facility is LOG_DAEMON.

-V, –dont-release-vrrp Don't remove VRRP VIPs and VROUTEs on daemon stop. The default behavior is to remove all VIPs and VROUTEs when keepalived exits

-I, –dont-release-ipvs Don't remove IPVS topology on daemon stop. The default behavior is to remove all entries from the IPVS virtual server table on when keepalived exits.

-R, –dont-respawn Don't respawn child processes. The default behavior is to restart the VRRP and checker processes if either process exits.

-n, –dont-fork Don't fork the daemon process. This option will cause keepalived to run in the foreground.

-d, –dump-conf Dump the configuration data.

-p, –pid=FILE Use specified pidfile for parent keepalived process. The default pidfile for keepalived is "/var/run/keepalived.pid".

-r, –vrrp_pid=FILE Use specified pidfile for VRRP child process. The default pidfile for the VRRP child process is "/var/run/keepalived_vrrp.pid".

-c, –checkers_pid=FILE Use specified pidfile for checkers child process. The default pidfile for the checker child process is "/var/run/keepalived_checkers.pid".

-x, –snmp Enable SNMP subsystem.

-v, –version Display the version and exit.

-h, –help Display this help message and exit.

6.2 genhash utility

The `genhash` binary is used to generate digest strings. The genhash command line arguments are:

–use-ssl, -S Use SSL to connect to the server.

–server <host>, -s Specify the ip address to connect to.

–port <port>, -p Specify the port to connect to.

–url <url>, -u Specify the path to the file you want to generate the hash of.

–use-virtualhost <host>, -V Specify the virtual host to send along with the HTTP headers.

–hash <alg>, -H Specify the hash algorithm to make a digest of the target page. Consult the help screen for list of available ones with a mark of the default one.

–verbose, -v Be verbose with the output.

–help, -h Display the program help screen and exit.

–release, -r Display the release number (version) and exit.

6.3 Running Keepalived daemon

To run Keepalived simply type:

```
[root@lvs tmp]# /etc/rc.d/init.d/keepalived.init start
Starting Keepalived for LVS:                                [ OK ]
```

All daemon messages are logged through the Linux syslog. If you start Keepalived with the "dump configuration data" option, you should see in your /var/log/messages (on Debian this may be */var/log/daemon.log* depending on your syslog configuration) something like this:

```
Jun 7 18:17:03 lvs1 Keepalived: Starting Keepalived v0.6.1 (06/13, 2002)
Jun 7 18:17:03 lvs1 Keepalived: Configuration is using : 92013 Bytes
Jun 7 18:17:03 lvs1 Keepalived: ------< Global definitions >------
Jun 7 18:17:03 lvs1 Keepalived: LVS ID = LVS_PROD
Jun 7 18:17:03 lvs1 Keepalived: Smtp server = 192.168.200.1
Jun 7 18:17:03 lvs1 Keepalived: Smtp server connection timeout = 30
Jun 7 18:17:03 lvs1 Keepalived: Email notification from = keepalived@domain.com
Jun 7 18:17:03 lvs1 Keepalived: Email notification = alert@domain.com
Jun 7 18:17:03 lvs1 Keepalived: Email notification = 0633556699@domain.com
Jun 7 18:17:03 lvs1 Keepalived: ------< SSL definitions >------
Jun 7 18:17:03 lvs1 Keepalived: Using autogen SSL context
Jun 7 18:17:03 lvs1 Keepalived: ------< LVS Topology >------
```

(continues on next page)

(continued from previous page)

```
Jun 7 18:17:03 lvs1 Keepalived: System is compiled with LVS v0.9.8
Jun 7 18:17:03 lvs1 Keepalived: VIP = 10.10.10.2, VPORT = 80
Jun 7 18:17:03 lvs1 Keepalived: VirtualHost = www.domain1.com
Jun 7 18:17:03 lvs1 Keepalived: delay_loop = 6, lb_algo = rr
Jun 7 18:17:03 lvs1 Keepalived: persistence timeout = 50
Jun 7 18:17:04 lvs1 Keepalived: persistence granularity = 255.255.240.0
Jun 7 18:17:04 lvs1 Keepalived: protocol = TCP
Jun 7 18:17:04 lvs1 Keepalived: lb_kind = NAT
Jun 7 18:17:04 lvs1 Keepalived: sorry server = 192.168.200.200:80
Jun 7 18:17:04 lvs1 Keepalived: RIP = 192.168.200.2, RPORT = 80, WEIGHT = 1
Jun 7 18:17:04 lvs1 Keepalived: RIP = 192.168.200.3, RPORT = 80, WEIGHT = 2
Jun 7 18:17:04 lvs1 Keepalived: VIP = 10.10.10.3, VPORT = 443
Jun 7 18:17:04 lvs1 Keepalived: VirtualHost = www.domain2.com
Jun 7 18:17:04 lvs1 Keepalived: delay_loop = 3, lb_algo = rr
Jun 7 18:17:04 lvs1 Keepalived: persistence timeout = 50
Jun 7 18:17:04 lvs1 Keepalived: protocol = TCP
Jun 7 18:17:04 lvs1 Keepalived: lb_kind = NAT
Jun 7 18:17:04 lvs1 Keepalived: RIP = 192.168.200.4, RPORT = 443, WEIGHT = 1
Jun 7 18:17:04 lvs1 Keepalived: RIP = 192.168.200.5, RPORT = 1358, WEIGHT = 1
Jun 7 18:17:05 lvs1 Keepalived: ------< Health checkers >------
Jun 7 18:17:05 lvs1 Keepalived: 192.168.200.2:80
Jun 7 18:17:05 lvs1 Keepalived: Keepalive method = HTTP_GET
Jun 7 18:17:05 lvs1 Keepalived: Connection timeout = 3
Jun 7 18:17:05 lvs1 Keepalived: Nb get retry = 3
Jun 7 18:17:05 lvs1 Keepalived: Delay before retry = 3
Jun 7 18:17:05 lvs1 Keepalived: Checked url = /testurl/test.jsp,
Jun 7 18:17:05 lvs1 Keepalived: digest = 640205b7b0fc66c1ea91c463fac6334d
Jun 7 18:17:05 lvs1 Keepalived: 192.168.200.3:80
Jun 7 18:17:05 lvs1 Keepalived: Keepalive method = HTTP_GET
Jun 7 18:17:05 lvs1 Keepalived: Connection timeout = 3
Jun 7 18:17:05 lvs1 Keepalived: Nb get retry = 3
Jun 7 18:17:05 lvs1 Keepalived: Delay before retry = 3
Jun 7 18:17:05 lvs1 Keepalived: Checked url = /testurl/test.jsp,
Jun 7 18:17:05 lvs1 Keepalived: digest = 640205b7b0fc66c1ea91c463fac6334c
Jun 7 18:17:05 lvs1 Keepalived: Checked url = /testurl2/test.jsp,
Jun 7 18:17:05 lvs1 Keepalived: digest = 640205b7b0fc66c1ea91c463fac6334c
Jun 7 18:17:06 lvs1 Keepalived: 192.168.200.4:443
Jun 7 18:17:06 lvs1 Keepalived: Keepalive method = SSL_GET
Jun 7 18:17:06 lvs1 Keepalived: Connection timeout = 3
Jun 7 18:17:06 lvs1 Keepalived: Nb get retry = 3
Jun 7 18:17:06 lvs1 Keepalived: Delay before retry = 3
Jun 7 18:17:06 lvs1 Keepalived: Checked url = /testurl/test.jsp,
Jun 7 18:17:05 lvs1 Keepalived: digest = 640205b7b0fc66c1ea91c463fac6334d
Jun 7 18:17:06 lvs1 Keepalived: Checked url = /testurl2/test.jsp,
Jun 7 18:17:05 lvs1 Keepalived: digest = 640205b7b0fc66c1ea91c463fac6334d
Jun 7 18:17:06 lvs1 Keepalived: 192.168.200.5:1358
Jun 7 18:17:06 lvs1 Keepalived: Keepalive method = TCP_CHECK
Jun 7 18:17:06 lvs1 Keepalived: Connection timeout = 3
Jun 7 18:17:06 lvs1 Keepalived: Registering Kernel netlink reflector
```

CHAPTER 7

IPVS Scheduling Algorithms

The following scheduling algorithms are supported by the IPVS kernel code. (heavily stolen from LVS website).

7.1 Round Robin (rr)

The round-robin scheduling algorithm sends each incoming request to the next server in its list. Thus in a three server cluster (servers A, B, and C) request 1 would go to server A, request 2 would go to server B, request 3 would go to server C, and request 4 would go to server A, thus completing the cycling or 'round-robin' of servers. It treats all real servers as equals regardless of the number of incoming connections or response time each server is experiencing. Virtual Server provides a few advantages over traditional round-robin DNS. Round-robin DNS resolves a single domain to the different IP addresses, the scheduling granularity is host-based, and the caching of DNS queries hinders the basic algorithm, these factors lead to significant dynamic load imbalances among the real servers. The scheduling granularity of Virtual Server is network connection-based, and it is much superior to round-robin DNS due to the fine scheduling granularity.

7.2 Weighted Round Robin (wrr)

The weighted round-robin scheduling is designed to better handle servers with different processing capacities. Each server can be assigned a weight, an integer value that indicates the processing capacity. Servers with higher weights receive new connections first than those with less weights, and servers with higher weights get more connections than those with less weights and servers with equal weights get equal connections. For example, the real servers, A, B, and C, have the weights, 4, 3, 2 respectively, a good scheduling sequence will be AABABCABC in a scheduling period (mod sum(Wi)). In the implementation of the weighted round-robin scheduling, a scheduling sequence will be generated according to the server weights after the rules of Virtual Server are modified. The network connections are directed to the different real servers based on the scheduling sequence in a round-robin manner.

The weighted round-robin scheduling is better than the round-robin scheduling, when the processing capacity of real servers is different. However, it may lead to dynamic load imbalance among the real servers if the load of the requests vary highly. In short, there is the possibility that a majority of requests requiring large responses may be directed to the same real server.

Actually, the round-robin scheduling is a special instance of the weighted round-robin scheduling, in which all the weights are equal.

7.3 Least Connection (lc)

The least-connection scheduling algorithm directs network connections to the server with the least number of established connections. This is one of the dynamic scheduling algorithms; because it needs to count live connections for each server dynamically. For a Virtual Server that is managing a collection of servers with similar performance, least-connection scheduling is good to smooth distribution when the load of requests varies a lot. Virtual Server will direct requests to the real server with the fewest active connections.

At a first glance, it might seem that least-connection scheduling can also perform well even when there are servers of various processing capacities, because the faster server will get more network connections. In fact, it cannot perform very well because of the TCP's TIME_WAIT state. The TCP's TIME_WAIT is usually 2 minutes, during this 2 minutes a busy website often receives thousands of connections, for example, the server A is twice as powerful as the server B, the server A is processing thousands of requests and keeping them in the TCP's TIME_WAIT state, but server B is crawling to get its thousands of connections finished. So, the least-connection scheduling cannot get load well balanced among servers with various processing capacities.

7.4 Weighted Least Connection (wlc)

The weighted least-connection scheduling is a superset of the least-connection scheduling, in which you can assign a performance weight to each real server. The servers with a higher weight value will receive a larger percentage of live connections at any one time. The Virtual Server Administrator can assign a weight to each real server, and network connections are scheduled to each server in which the percentage of the current number of live connections for each server is a ratio to its weight. The default weight is one.

The weighted least-connections scheduling works as follows:

> Supposing there are n real servers, each server i has weight Wi (i=1,..,n), and alive connections Ci (i=1,..,n), ALL_CONNECTIONS is the sum of Ci (i=1,..,n), the next network connection will be directed to the server j, in which
>
> > (Cj/ALL_CONNECTIONS)/Wj = min { (Ci/ALL_CONNECTIONS)/Wi } (i=1,..,n)
>
> Since the ALL_CONNECTIONS is a constant in this lookup, there is no need to divide Ci by ALL_CONNECTIONS, it can be optimized as
>
> > Cj/Wj = min { Ci/Wi } (i=1,..,n)

The weighted least-connection scheduling algorithm requires additional division than the least-connection. In a hope to minimize the overhead of scheduling when servers have the same processing capacity, both the least-connection scheduling and the weighted least-connection scheduling algorithms are implemented.

7.5 Locality-Based Least Connection (lblc)

The locality-based least-connection scheduling algorithm is for destination IP load balancing. It is usually used in cache cluster. This algorithm usually directs packet destined for an IP address to its server if the server is alive and under load. If the server is overloaded (its active connection numbers is larger than its weight) and there is a server in its half load, then allocate the weighted least-connection server to this IP address.

7.6 Locality-Based Least Connection with Replication (lblcr)

The locality-based least-connection with replication scheduling algorithm is also for destination IP load balancing. It is usually used in cache cluster. It differs from the LBLC scheduling as follows: the load balancer maintains mappings from a target to a set of server nodes that can serve the target. Requests for a target are assigned to the least-connection node in the target's server set. If all the node in the server set are over loaded, it picks up a least-connection node in the cluster and adds it in the sever set for the target. If the server set has not been modified for the specified time, the most loaded node is removed from the server set, in order to avoid high degree of replication.

7.7 Destination Hashing (dh)

The destination hashing scheduling algorithm assigns network connections to the servers through looking up a statically assigned hash table by their destination IP addresses.

7.8 Source Hashing (sh)

The source hashing scheduling algorithm assigns network connections to the servers through looking up a statically assigned hash table by their source IP addresses.

7.9 Shortest Expected Delay (sed)

The shortest expected delay scheduling algorithm assigns network connections to the server with the shortest expected delay. The expected delay that the job will experience is (Ci + 1) / Ui if sent to the ith server, in which Ci is the number of connections on the ith server and Ui is the fixed service rate (weight) of the ith server.

7.10 Never Queue (nq)

The never queue scheduling algorithm adopts a two-speed model. When there is an idle server available, the job will be sent to the idle server, instead of waiting for a fast one. When there is no idle server available, the job will be sent to the server that minimizes its expected delay (The Shortest Expected Delay scheduling algorithm).

7.11 Overflow-Connection (ovf)

The Overflow connection scheduling algorithm implements "overflow" loadbalancing according to a number of active connections, will keep all connections to the node with the highest weight and overflow to the next node if the number of connections exceeds the node's weight. Note that this scheduler might not be suitable for UDP because it only uses active connections

CHAPTER 8

IPVS Protocol Support

8.1 IPVS Transport Protocol Load Balancing Support

IPVS supports load balancing the following transport protocols:

- TCP
- UDP
- SCTP
- ESP (Encapsulation Security Payload)
- AH (Authentication Header)

8.2 IPVS Application Helpers

8.2.1 FTP Protocol Helper

8.2.2 Netfilter Connection Tracking

8.2.3 SIP Persistence Engine

Depends on UDP and SIP netfilter connection tracking.

Todo: Mention Netfilter Connection Tracking

Todo: Mention ipip kernel module

CHAPTER 9

Configuring SNMP Support

Keepalived provides an SNMP subsystem that can gather various metrics about the VRRP stack and the health checker system. The keepalived MIB is in the `doc` directory of the project. The base SNMP OID for the MIB is *.1.3.6.1.4.1.9586.100.5*, which is hosted under the Debian OID space assigned by IANA.

Prerequisites *********

Install the SNMP protocol tools and libraries onto your system. This requires the installation of a few packages:

```
yum install net-snmp net-snmp-utils net-snmp-libs
```

Once SNMP has been installed on your system, configure keepalived with SNMP support. When compiling keepalived, add the `--enable-snmp` configure option. For example:

```
./configure --enable-snmp
```

During the configure step of the compiling process, you will get a configuration summary before building with `make`. For example, you may see similar output on a CentOS 6 machine:

```
./configure --prefix=/usr/local/keepalived-1.2.15 --enable-snmp
Keepalived configuration
------------------------
Keepalived version       : 1.2.15
Compiler                 : gcc
Compiler flags           : -g -O2 -I/usr/include/libnl3
Extra Lib                : -Wl,-z,relro -Wl,-z,now -L/usr/lib64
-lnetsnmpagent -lnetsnmphelpers -lnetsnmpmibs -lnetsnmp -Wl,-E
-Wl,-rpath,/usr/lib64/perl5/CORE -lssl -lcrypto -lcrypt  -lnl-genl-3 -lnl-3
Use IPVS Framework       : Yes
IPVS sync daemon support : Yes
IPVS use libnl           : Yes
fwmark socket support    : Yes
Use VRRP Framework       : Yes
Use VRRP VMAC            : Yes
SNMP support             : Yes
```

(continues on next page)

(continued from previous page)

```
SHA1 support                : No
Use Debug flags             : No
```

Notice the *Extra Lib* section of the configuration summary. It lists various library flags that gcc will use to build keepalived, several of which have to do with SNMP.

9.1 Configuring Support

Enable SNMP AgentX support by including the following line in the SNMP daemon configuration file, typically `/etc/snmp/snmpd.conf` if you installed via RPMs on a CentOS machine:

```
master agentx
```

Note: Be sure to reload or restart the SNMP service for the configuration change to take effect.

9.2 Adding the MIB

You can query keepalived SNMP managed objects by using the OID. For example:

```
snmpwalk -v2c -c public localhost .1.3.6.1.4.1.9586.100.5.1.1.0
SNMPv2-SMI::enterprises.9586.100.5.1.1.0 = STRING: "Keepalived v1.2.15 (01/10,2015)"
```

Alternatively, with the keepalived MIB, you can query using the MIB available from the project. First, copy the MIB to the system's global MIB directory or to the user's local MIB directory:

```
cp /usr/local/src/keepalived-1.2.15/doc/KEEPALIVED-MIB /usr/share/snmp/mibs
```

or:

```
cp /usr/local/src/keepalived-1.2.15/doc/KEEPALIVED-MIB ~/.snmp/mibs
```

The SNMP daemon will check both directories for the existence of the MIB. Once the MIB is in place, the SNMP query can look as follows:

```
snmpwalk -v2c -c public localhost KEEPALIVED-MIB::version
KEEPALIVED-MIB::version.0 = STRING: Keepalived v1.2.15 (01/10,2015)
```

9.3 MIB Overview

There are four main sections to the keepalived MIB:

- global
- vrrp
- check
- conformance

9.3.1 Global

The global section includes objects that contain information about the keepalived instance such as version, router ID and administrative email addresses.

9.3.2 VRRP

The VRRP section includes objects that contain information about each configured VRRP instance. Within each instance, there are objects that include instance name, current state, and virtual IP addresses.

9.3.3 Check

The Check section includes objects that contain information about each configured virtual server. It includes server tables for virtual and real servers and also configured load balancing algorithms, load balancing method, protocol, status, real and virtual server network connection statistics.

9.3.4 Conformance

Todo: do conformance

Note: Use a MIB browser, such as mbrowse, to see what managed objects are available to query for monitoring the health of your LVS servers.

CHAPTER 10

Case Study: Healthcheck

As an example we can introduce the following LVS topology:

First of all, you need a well-configured LVS topology. In the rest of this document, we will assume that all system configurations have been done. This kind of topology is generally implemented in a DMZ architecture. For more information on LVS NAT topology and system configuration please read the nice Joseph Mack LVS HOWTO.

10.1 Main architecture components

- LVS Router: Owning the load balanced IP Class routed (192.168.100.0/24).
- Network Router: The default router for the entire internal network. All the LAN workstations are handled through this IP address.
- Network DNS Server: Referencing the internal network IP topology.
- SMTP Server: SMTP server receiving the mail alerts.
- SERVER POOL: Set of servers hosting load balanced services.

10.2 Server pool specifications

In this sample configuration we have 2 server pools:

- Server pool 1: Hosting the HTTP & SSL services. Each server owns two application servers (IBM WEBSPHERE & BEA WEBLOGIC)
- Server pool 2: Hosting the SMTP service.

10.3 Keepalived configuration

You are now ready to configure the Keepalived daemon according to your LVS topology. The whole configuration is done in the /etc/keepalived/keepalived.conf file. In our case study this file looks like:

```
# Configuration File for keepalived
global_defs {
    notification_email {
        admin@domain.com
        0633225522@domain.com
    }
    notification_email_from keepalived@domain.com
    smtp_server 192.168.200.20
    smtp_connect_timeout 30
    lvs_id LVS_MAIN
}
virtual_server 192.168.200.15 80 {
    delay_loop 30
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    sorry_server 192.168.100.100 80

    real_server 192.168.100.2 80 {
        weight 2
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest ec90a42b99ea9a2f5ecbe213ac9eba03
            }
            url {
                path /testurl2/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            retry 3
            delay_before_retry 2
        }
    }
    real_server 192.168.100.3 80 {
        weight 1
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            retry 3
            delay_before_retry 2
        }
    }
}
virtual_server 192.168.200.15 443 {
    delay_loop 20
    lb_algo rr
```

(continues on next page)

(continued from previous page)

```
    lb_kind NAT
    persistence_timeout 360
    protocol TCP
    real_server 192.168.100.2 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.100.3 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
virtual_server 192.168.200.15 25 {
    delay_loop 15
    lb_algo wlc
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    real_server 192.168.100.4 25 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.100.5 25 {
        weight 2
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
```

According to this configuration example, the Keepalived daemon will drive the kernel using the following information:

- The LVS server will own the name: LVS_MAIN
- Notification:
  - SMTP server will be: 192.168.200.20
  - SMTP connection timeout is set to: 30 seconded
  - Notification emails will be: admin@domain.com & 0633225522@domain.com
- Load balanced services:
  - HTTP: VIP 192.168.200.15 port 80
    - Load balancing: Using Weighted Round Robin scheduler with NAT forwarding. Connection persistence is set to 50 seconds on each TCP service. If you are using Linux kernel 2.2 you need to specify the NAT netmask to define the IPFW masquerade granularity (nat_mask keyword). The delay loop is set to 30 seconds
    - Sorry Server: If all real servers are removed from the VS's server pools, we add the sorry_server 192.168.100.100 port 80 to serve clients requests.

        - Real server 192.168.100.2 port 80 will be weighted to 2. Failure detection will be based on HTTP_GET over 2 URLS. The service connection timeout will be set to 3 seconds. The real server will be considered down after 3 retries. The daemon will wait for 2 seconds before retrying.
        - Real server 192.168.100.3 port 80 will be weighted to 1. Failure detection will be based on HTTP_GET over 1 URL. The service connection timeout will be set to 3 seconds. The real server will be considered down after 3 retries. The daemon will wait for 2 seconds before retrying.
    - SSL: VIP 192.168.200.15 port 443
        - Load balancing: Using Round Robin scheduler with NAT forwarding. Connection persistence is set to 360 seconds on each TCP service. The delay loop is set to 20 seconds
        - Real server 192.168.100.2 port 443 will be weighted to 2. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3-second connection timeout.
        - Real server 192.168.100.3 port 443 will be weighted to 2. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3-second connection timeout.
    - SMTP: VIP 192.168.200.15 port 25
        - Load balancing: Using Weighted Least Connection scheduling algorithm in a NAT topology with connection persistence set to 50 seconds. The delay loop is set to 15 seconds
        - Real server 192.168.100.4 port 25 will be weighted to 1. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3-second connection timeout.
        - Real server 192.168.100.5 port 25 will be weighted to 2. Failure detection will be based on TCP_CHECK. The real server will be considered down after a 3-second connection timeout.

For SSL server health check, we can use SSL_GET checkers. The configuration block for a corresponding real server will look like:

```
virtual_server 192.168.200.15 443 {
    delay_loop 20
    lb_algo rr
    lb_kind NAT
    persistence_timeout 360
    protocol TCP
    real_server 192.168.100.2 443 {
        weight 1
        SSL_GET
        {
            url {
                path /testurl/test.jsp
                digest ec90a42b99ea9a2f5ecbe213ac9eba03
            }
            url {
                path /testurl2/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            retry 3
            delay_before_retry 2
        }
    }
    real_server 192.168.100.3 443 {
        weight 1
        SSL_GET
        {
            url {
```

(continues on next page)

(continued from previous page)

```
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            retry 3
            delay_before_retry 2
        }
    }
}
```

To generate a sum over an URL simply proceed as follows:

```
[root@lvs /root]# genhash -s 192.168.100.2 -p 80 -u /testurl/test.jsp
--------------------------[ HTTP Header Buffer ]--------------------------
0000 48 54 54 50 2f 31 2e 31 - 20 34 30 31 20 55 6e 61 HTTP/1.1 401 Una
0010 75 74 68 6f 72 69 7a 65 - 64 0d 0a 44 61 74 65 3a uthorized..Date:
0020 20 4d 6f 6e 2c 20 32 33 - 20 41 70 72 20 32 30 30 Mon, 23 Apr 200
0030 31 20 31 35 3a 34 31 3a - 35 34 20 47 4d 54 0d 0a 1 15:41:54 GMT..
0040 41 6c 6c 6f 77 3a 20 47 - 45 54 2c 20 48 45 41 44 Allow: GET, HEAD
0050 0d 0a 53 65 72 76 65 72 - 3a 20 4f 72 61 63 6c 65 ..Server: Oracle
0060 5f 57 65 62 5f 4c 69 73 - 74 65 6e 65 72 2f 34 2e _Web_Listener/4.
0070 30 2e 38 2e 31 2e 30 45 - 6e 74 65 72 70 72 69 73 0.8.1.0Enterpris
0080 65 45 64 69 74 69 6f 6e - 0d 0a 43 6f 6e 74 65 6e eEdition..Conten
0090 74 2d 54 79 70 65 3a 20 - 74 65 78 74 2f 68 74 6d t-Type: text/htm
00a0 6c 0d 0a 43 6f 6e 74 65 - 6e 74 2d 4c 65 6e 67 74 l..Content-Lengt
00b0 68 3a 20 31 36 34 0d 0a - 57 57 57 2d 41 75 74 68 h: 164..WWW-Auth
00c0 65 6e 74 69 63 61 74 65 - 3a 20 42 61 73 69 63 20 enticate: Basic
00d0 72 65 61 6c 6d 3d 22 41 - 43 43 45 53 20 20 20 20 realm="ACCES
00e0 22 0d 0a 43 61 63 68 65 - 2d 43 6f 6e 74 72 6f 6c "..Cache-Control
00f0 3a 20 70 75 62 6c 69 63 - 0d 0a 0d 0a : public....
------------------------------[ HTML Buffer ]-----------------------------
0000 3c 48 54 4d 4c 3e 3c 48 - 45 41 44 3e 3c 54 49 54 <HTML><HEAD><TIT
0010 4c 45 3e 55 6e 61 75 74 - 68 6f 72 69 7a 65 64 3c LE>Unauthorized<
0020 2f 54 49 54 4c 45 3e 3c - 2f 48 45 41 44 3e 0d 0a /TITLE></HEAD>..
0030 3c 42 4f 44 59 3e 54 68 - 69 73 20 64 6f 63 75 6d <BODY>This docum
0040 65 6e 74 20 69 73 20 70 - 72 6f 74 65 63 74 65 64 ent is protected
0050 2e 20 20 59 6f 75 20 6d - 75 73 74 20 73 65 6e 64 . You must send
0060 0d 0a 74 68 65 20 70 72 - 6f 70 65 72 20 61 75 74 ..the proper aut
0070 68 6f 72 69 7a 61 74 69 - 6f 6e 20 69 6e 66 6f 72 horization infor
0080 6d 61 74 69 6f 6e 20 74 - 6f 20 61 63 63 65 73 73 mation to access
0090 20 69 74 2e 3c 2f 42 4f - 44 59 3e 3c 2f 48 54 4d it.</BODY></HTM
00a0 4c 3e 0d 0a - L>..
-----------------------[ HTML MD5 final resulting ]-----------------------
MD5 Digest : ec90a42b99ea9a2f5ecbe213ac9eba03
```

The only thing to do is to copy the generated MD5 Digest value generated and paste it into your Keepalived configuration file as a digest value keyword.

CHAPTER 11

Case Study: Failover using VRRP

As an example we can introduce the following LVS topology:

11.1 Architecture Specification

To create a virtual LVS director using the VRRPv2 protocol, we define the following architecture:

- 2 LVS directors in active-active configuration.
- 4 VRRP Instances per LVS director: 2 VRRP Instance in the MASTER state and 2 in BACKUP state. We use a symmetric state on each LVS directors.
- 2 VRRP Instances in the same state are to be synchronized to define a persistent virtual routing path.
- Strong authentication: IPSEC-AH is used to protect our VRRP advertisements from spoofed and reply attacks.

The VRRP Instances are compounded with the following IP addresses:

- VRRP Instance VI_1: owning VRRIP VIPs VIP1 & VIP2. This instance defaults to the MASTER state on LVS director 1. It stays synchronized with VI_2.
- VRRP Instance VI_2: owning DIP1. This instance is by default in MASTER state on LVS director 1. It stays synchronized with VI_1.
- VRRP Instance VI_3: owning VRRIP VIPs VIP3 & VIP4. This instance is in default MASTER state on LVS director 2. It stays synchronized with VI_4.
- VRRP Instance VI_4: owning DIP2. This instance is in default MASTER state on LVS director 2. It stays synchronized with VI_3.

11.2 Keepalived Configuration

The whole configuration is done in the /etc/keepalived/keepalived.conf file. In our case study this file on LVS director 1 looks like:

```
vrrp_sync_group VG1 {
    group {
        VI_1
        VI_2
    }
}
vrrp_sync_group VG2 {
    group {
        VI_3
        VI_4
    }
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve1
    }
    virtual_ipaddress {
        192.168.200.10
        192.168.200.11
    }
}
vrrp_instance VI_2 {
    state MASTER
    interface eth1
    virtual_router_id 52
    priority 150
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve2
    }
    virtual_ipaddress {
        192.168.100.10
    }
}
```

```
vrrp_instance VI_3 {
    state BACKUP
    interface eth0
    virtual_router_id 53
    priority 100
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve3
    }
    virtual_ipaddress {
        192.168.200.12
        192.168.200.13
    }
}
```

(continues on next page)

(continued from previous page)

```
vrrp_instance VI_4 {
    state BACKUP
    interface eth1
    virtual_router_id 54
    priority 100
    advert_int 1
    authentication {
        auth_type AH
        auth_pass k@l!ve4
    }
    virtual_ipaddress {
        192.168.100.11
    }
}
```

Then we define the symmetric configuration file on LVS director 2. This means that VI_3 & VI_4 on LVS director 2 are in MASTER state with a higher priority 150 to start with a stable state. Symmetrically VI_1 & VI_2 on LVS director 2 are in default BACKUP state with lower priority of 100. | This configuration file specifies 2 VRRP Instances per physical NIC. When you run Keepalived on LVS director 1 without running it on LVS director 2, LVS director 1 will own all the VRRP VIP. So if you use the ip utility you may see something like: (On Debian the ip utility is part of iproute):

```
[root@lvs1 tmp]# ip address list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.5/24 brd 192.168.200.255 scope global eth0
    inet 192.168.200.10/32 scope global eth0
    inet 192.168.200.11/32 scope global eth0
    inet 192.168.200.12/32 scope global eth0
    inet 192.168.200.13/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.201.255 scope global eth1
    inet 192.168.100.10/32 scope global eth1
    inet 192.168.100.11/32 scope global eth1
```

Then simply start Keepalived on the LVS director 2 and you will see:

```
[root@lvs1 tmp]# ip address list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.5/24 brd 192.168.200.255 scope global eth0
    inet 192.168.200.10/32 scope global eth0
    inet 192.168.200.11/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.201.255 scope global eth1
    inet 192.168.100.10/32 scope global eth1
```

Symmetrically on LVS director 2 you will see:

```
[root@lvs2 tmp]# ip address list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:10 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.5/24 brd 192.168.200.255 scope global eth0
    inet 192.168.200.12/32 scope global eth0
    inet 192.168.200.13/32 scope global eth0
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:00:5e:00:01:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.201.255 scope global eth1
    inet 192.168.100.11/32 scope global eth1
```

The VRRP VIPs are:

- VIP1 = 192.168.200.10
- VIP2 = 192.168.200.11
- VIP3 = 192.168.200.12
- VIP4 = 192.168.200.13
- DIP1 = 192.168.100.10
- DIP2 = 192.168.100.11

The use of VRRP keyword “sync_instance” imply that we have defined a pair of MASTER VRRP Instance per LVS directors ó (VI_1,VI_2) & (VI_3,VI_4). This means that if eth0 on LVS director 1 fails then VI_1 enters the MASTER state on LVS director 2 so the MASTER Instance distribution on both directors will be: (VI_2) on director 1 & (VI_1,VI_3,VI_4) on director 2. We use “sync_instance” so VI_2 is forced to BACKUP the state on LVS director 1. The final VRRP MASTER instance distribution will be: (none) on LVS director 1 & (VI_1,VI_2,VI_3,VI_4) on LVS director 2. If eth0 on LVS director 1 became available the distribution will transition back to the initial state.

For more details on this state transition please refer to the “Linux Virtual Server High Availability using VRRPv2” paper (available at http://www.linux-vs.org/~acassen/), which explains the implementation of this functionality.

Using this configuration both LVS directors are active at a time, thus sharing LVS directors for a global director. That way we introduce a virtual LVS director.

Note: This VRRP configuration sample is an illustration for a high availability router (not LVS specific). It can be used for many more common/simple needs.

CHAPTER 12

Case Study: Mixing Healthcheck & Failover

For this example, we use the same topology used in the Failover part. The idea here is to use VRRP VIPs as LVS VIPs. That way we will introduce a High Available LVS director performing LVS real server pool monitoring.

12.1 Keepalived Configuration

The whole configuration is done in the /etc/keepalived/keepalived.conf file. In our case study this file on LVS director 1 looks like:

```
# Configuration File for keepalived
global_defs {
    notification_email {
        admin@domain.com
        0633225522@domain.com
    }
    notification_email_from keepalived@domain.com
    smtp_server 192.168.200.20
    smtp_connect_timeout 30
    lvs_id LVS_MAIN
}
# VRRP Instances definitions
vrrp_sync_group VG1 {
    group {
        VI_1
        VI_2
    }
}
vrrp_sync_group VG2 {
    group {
        VI_3
        VI_4
    }
}
```

(continues on next page)

(continued from previous page)

```
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve1
    }
    virtual_ipaddress {
        192.168.200.10
        192.168.200.11
    }
}
vrrp_instance VI_2 {
    state MASTER
    interface eth1
    virtual_router_id 52
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve2
    }
    virtual_ipaddress {
        192.168.100.10
    }
}
vrrp_instance VI_3 {
    state BACKUP
    interface eth0
    virtual_router_id 53
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve3
    }
    virtual_ipaddress {
        192.168.200.12
        192.168.200.13
    }
}
vrrp_instance VI_4 {
    state BACKUP
    interface eth1
    virtual_router_id 54
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass k@l!ve4
    }
    virtual_ipaddress {
        192.168.100.11
    }
```

(continues on next page)

(continued from previous page)

```
}
# Virtual Servers definitions
virtual_server 192.168.200.10 80 {
    delay_loop 30
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    sorry_server 192.168.100.100 80
    real_server 192.168.100.2 80 {
        weight 2
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest ec90a42b99ea9a2f5ecbe213ac9eba03
            }
            url {
                path /testurl2/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            retry 3
            delay_before_retry 2
        }
    }
    real_server 192.168.100.3 80 {
        weight 1
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334c
            }
            connect_timeout 3
            retry 3
            delay_before_retry 2
        }
    }
}
virtual_server 192.168.200.12 443 {
    delay_loop 20
    lb_algo rr
    lb_kind NAT
    persistence_timeout 360
    protocol TCP
    real_server 192.168.100.2 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.100.3 443 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
```

We define the symmetric VRRP configuration file on LVS director 2. That way both directors are active at a time, director 1 handling HTTP stream and director 2 SSL stream.

CHAPTER 13

Terminology

Todo: put image here

LVS stands for “Linux Virtual Server“. LVS is a patched Linux kernel that adds a load balancing facility. For more information on LVS, please refer to the project homepage: http://www.linux-vs.org. LVS acts as a network bridge (using NAT) to load balance TCP/UDP stream. The LVS router components are:

- WAN Interface: Ethernet Network Interface Controller that will be accessed by all the clients.
- LAN Interface: Ethernet Network Interface Controller to manage all the load balanced servers.
- Linux kernel: The kernel is patched with the latest LVS and is used as a router OS.

In this document, we will use the following keywords:

13.1 LVS Component

VIP The Virtual IP is the IP address that will be accessed by all the clients. The clients only access this IP address.

Real server A real server hosts the application accessed by client requests. WEB SERVER 1 & WEB SERVER 2 in our synopsis.

Server pool A farm of real servers.

Virtual server The access point to a Server pool.

Virtual Service A TCP/UDP service associated with the VIP.

13.2 VRRP Component

VRRP The protocol implemented for the directors’ failover/virtualization.

IP Address owner The VRRP Instance that has the IP address(es) as real interface address(es). This is the VRRP Instance that, when up, will respond to packets addressed to one of these IP address(es) for ICMP, TCP connections, …

MASTER state VRRP Instance state when it is assuming the responsibility of forwarding packets sent to the IP address(es) associated with the VRRP Instance. This state is illustrated on "Case study: Failover" by a red line.

BACKUP state VRRP Instance state when it is capable of forwarding packets in the event that the current VRRP Instance MASTER fails.

Real Load Balancer An LVS director running one or many VRRP Instances.

Virtual Load balancer A set of Real Load balancers.

Synchronized Instance VRRP Instance with which we want to be synchronized. This provides VRRP Instance monitoring.

Advertisement The name of a simple VRRPv2 packet sent to a set of VRRP Instances while in the MASTER state.

Todo: Define RIP, DIP, Director, IPVS

CHAPTER 14

License

CHAPTER 15

About These Documents

These documents are generated from reStructuredText sources by Sphinx, a document processor specifically written for the Python documentation.

15.1 Building The Documentation

To build the keepalived documentation, you will need to have a recent version of Sphinx installed on your system. Alternatively, you could use a python virtualenv.

From the root of the repository clone, run the following command to build the documentation in HTML format:

```
cd keepalived-docs
make html
```

For PDF, you will also need `docutils` and various `texlive-*` packages for converting reStructuredText to LaTex and finally to PDF:

```
pip install docutils
cd keepalived-docs
make latexpdf
```

Alternatively, you can use the `sphinx-build` command that comes with the Sphinx package:

```
cd keepalived-docs
sphinx-build -b html . build/html
```

Todo: make latexpdf needs pdflatex provided by texlive-latex on RHEL6 and texlive-latex-bin-bin on Fedora21

Todo: make linkcheck to check for broken links

CHAPTER 16

TODO List

Todo: make latexpdf needs pdflatex provided by texlive-latex on RHEL6 and texlive-latex-bin-bin on Fedora21

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/about.rst, line 38.)

Todo: make linkcheck to check for broken links

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/about.rst, line 42.)

Todo: Mention Netfilter Connection Tracking

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/protocol_support.rst, line 32.)

Todo: Mention ipip kernel module

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/protocol_support.rst, line 35.)

Todo: do conformance

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/snmp_support.rst, line 127.)

Todo: put image here

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/terminology.rst, line 5.)

Todo: Define RIP, DIP, Director, IPVS

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/keepalived-pqa/checkouts/stable/doc/source/terminology.rst, line 74.)

Index