Abstract

Oracle® Linux 8: Setting Up Load Balancing describes tasks for setting up load balancing by using the Keepalived and HAProxy technologies.

Document generated on: 2020-02-12 (revision: 9257)
# Table of Contents

Preface .................................................................................................................................................. v
1 About Load Balancing in Oracle Linux ................................................................................................. 1
  1.1 About Load Balancing ...................................................................................................................... 1
  1.2 About HAProxy ................................................................................................................................ 1
  1.3 About Keepalived ............................................................................................................................ 1
  1.4 About Combining Keepalived With HAProxy for High-Availability Load Balancing .................. 2
  1.5 About NGINX ................................................................................................................................... 2
2 Setting Up Load Balancing by Using HAProxy ...................................................................................... 5
  2.1 Installing and Configuring HAProxy .................................................................................................. 5
  2.2 HAProxy Configuration Directives .................................................................................................. 5
  2.3 Configuring Round Robin Load Balancing by Using HAProxy ....................................................... 6
  2.4 Using Weighted Round Robin Load Balancing with HAProxy ........................................................ 7
  2.5 Adding Session Persistence for HAProxy .......................................................................................... 8
3 Setting Up Load Balancing by Using Keepalived ................................................................................ 9
  3.1 Installing and Configuring Keepalived ............................................................................................. 9
  3.2 Keepalived Configuration Directives ............................................................................................. 9
  3.3 Setting Up Load Balancing in NAT Mode by Using Keepalived .................................................... 10
    3.3.1 Configuring Firewall Rules for Keepalived NAT-Mode Load Balancing ................................. 14
    3.3.2 Configuring Back-End Server Routing for Keepalived NAT-Mode Load Balancing ............ 15
  3.4 Setting Up Load Balancing by Using Keepalived With HAProxy ................................................ 15
4 Setting Up Load Balancing by Using NGINX ....................................................................................... 19
  4.1 Installing NGINX ............................................................................................................................ 19
  4.2 NGINX Configuration Directives .................................................................................................... 19
  4.3 Configuring Round Robin Load Balancing by Using NGINX ........................................................ 21
  4.4 Using Weighted Round Robin Load Balancing With NGINX ....................................................... 22
  4.5 Using Least-Connected Load Balancing With NGINX ................................................................. 22
  4.6 Adding Session Persistence for NGINX ............................................................................................ 23
Preface

Oracle® Linux 8: Setting Up Load Balancing describes how to configure the Keepalived and HAProxy load balancer technologies for balanced access to network services.

Audience

This document is intended for administrators who need to configure and administer Oracle Linux networking. It is assumed that readers are familiar with web technologies and have a general understanding of using the Linux operating system, including knowledge of how to use a text editor such as emacs or vim, essential commands such as cd, chmod, chown, ls, mkdir, mv, ps, pwd, and rm, and using the man command to view manual pages.

Document Organization

The document is organized into the following chapters:

• Chapter 1, About Load Balancing in Oracle Linux describes load balancing, how it works, and its benefits, as well as the various load balancer technologies that are provided in Oracle Linux. Getting started installation and configuration information is also covered in this chapter.

• Chapter 2, Setting Up Load Balancing by Using HAProxy describes how to set up load balancing by using the HAProxy feature.

• Chapter 3, Setting Up Load Balancing by Using Keepalived describes how to set up load balancing by using the Keepalived feature. This chapter also provides an example that combines both Keepalived and HAProxy for load balancing.

• Chapter 4, Setting Up Load Balancing by Using NGINX describes how set up a basic load balancing by using NGINX.

Related Documents

The documentation for this product is available at:

Oracle® Linux 8 Documentation

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.
Chapter 1 About Load Balancing in Oracle Linux

Table of Contents
1.1 About Load Balancing ................................................................. 1
1.2 About HAProxy ................................................................. 1
1.3 About Keepalived ............................................................. 1
1.4 About Combining Keepalived With HAProxy for High-Availability Load Balancing ............... 2
1.5 About NGINX ................................................................. 2

This chapter provides an overview of the load balancer technologies that are used in Oracle Linux. Installation and configuration information is also provided in this chapter.

1.1 About Load Balancing

The term **load balancing** refers to the efficient distribution of incoming network traffic across a group of back-end servers. The use of load balancing ensures that your infrastructure is highly available, reliable, and that performance is not degraded. Load balancers can typically handle traffic for the HTTP, HTTPS, TCP, and UDP protocols.

Load balancers manage network traffic by routing client requests across all of the servers that can fulfill those requests. This routing maximizes speed and capacity utilization so that no one particular server becomes overloaded, thereby improving overall performance. In situations where a server may become unavailable or goes down, the load balancer redirects any incoming traffic to other servers that are online. In this way, server downtime is minimized. When a new server is added to the server group, the load balancer automatically redistributes the workload and starts to send requests to that new server.

In Oracle Linux, load balancing of network traffic is primarily handled by two integrated software components: HAProxy and Keepalived. The HAProxy feature provides load balancing and high-availability services to TCP and HTTP, while Keepalived performs load balancing and failover tasks on both active and passive routers. The NGINX feature can also be used in Oracle Linux for load balancing.

1.2 About HAProxy

HAProxy, or High Availability Proxy, is an application layer (Layer 7) load balancer and high-availability solution that you can use to implement a reverse proxy for HTTP and TCP-based Internet services. An application layer load balancer often includes many features, because it is able to inspect the content of the traffic that it is routing and can either modify content within each packet, as required, or can make decisions about how to handle each packet based on its content. This makes it simple to implement session persistence, TLS, ACLs, and HTTP rewrites and redirection.

The configuration file for the haproxy daemon is /etc/haproxy/haproxy.cfg. This file must be present on each server on which you configure HAProxy for load balancing or high availability.

For more information, see http://www.haproxy.org/#docs, the /usr/share/doc/haproxy-version documentation, and the haproxy(1) manual page.

1.3 About Keepalived

Keepalived uses the IP Virtual Server (IPVS) kernel module to provide transport layer (Layer 4) load balancing by redirecting requests for network-based services to individual members of a server cluster.
Using Keepalived With VRRP

IPVS monitors the status of each server and uses the Virtual Router Redundancy Protocol (VRRP) to implement high availability. A load balancer that functions at the transport layer is less aware of the content of the packets that it re-routes, which has the advantage of being able to perform this task significantly faster than a reverse proxy system functioning at the application layer.

The configuration file for the keepalived daemon is /etc/keepalived/keepalived.conf. This file must be present on each server on which you configure Keepalived for load balancing or high availability.

For more information, see http://www.keepalived.org/documentation.html, the /usr/share/doc/keepalive-version documentation, and the keepalived(8) and keepalived.conf(5) manual pages.

Using Keepalived With VRRP

VRRP is a networking protocol that automatically assigns routers that are available to handle inbound traffic. A detailed standard document for this protocol can be found at https://tools.ietf.org/html/rfc5798.

Keepalived uses VRRP to ascertain the current state of all of the routers on the network. The protocol enables routing to switch between master and back-up routers automatically. The back-up routers detect when the master router becomes unavailable and then sends multicast packets to each other until one of the routers is "elected" as the new master. A floating virtual IP address can be used to always direct traffic to the master router. When the original master router is back online, it detects the new routing state and returns to the network as a back-up router.

The benefit of using VRRP is that you can rely on multiple routers to provide high availability and redundancy without requiring a separate software service or hardware device to manage this process. On each router, Keepalived configures the VRRP settings and ensures that the network routing continues to function correctly.

For more information, see http://www.keepalived.org/documentation.html, the /usr/share/doc/keepalive-version documentation, and the keepalived(8) and keepalived.conf(5) manual pages.

1.4 About Combining Keepalived With HAProxy for High-Availability Load Balancing

You can combine the Keepalived and HAProxy load balancer features to achieve a high-availability, load-balancing environment. HAProxy provides scalability, application-aware functionality, and ease of configuration when configuring load balancing services. Keepalived provides failover services for back-up routers, as well as the ability to distribute loads across servers for increased availability.

This complex configuration scenario illustrates how you can use different load balancing applications with each other to achieve better redundancy and take advantage of features at different layers of the stack. While this example shows how Keepalived can be used to provide redundancy for HAProxy, you can also achieve similar results by using Keepalived with alternate application layer proxy systems, like NGINX.

For more details, see Section 3.4, "Setting Up Load Balancing by Using Keepalived With HAProxy".

1.5 About NGINX

NGINX is a well-known HTTP server that provides modular functionality for reverse proxing, traffic routing, and application-layer load balancing for HTTP, HTTPS or TCP/UDP connections. You can use NGINX load balancing and proxy services to distribute traffic for improved performance, scalability, and reliability of your applications.
NGINX provides capability for the following load balancing methods:

- **Round Robin.** This method is one of the simplest for implementing load balancing and is the default method that is used by NGINX. Round Robin distributes requests to application servers by going down the list of the servers that are within the group, then forwarding client requests to each server, in turn. After reaching the end of the list, the load balancer repeats this same sequence.

- **Least Connected.** This method works by assigning the next request to the server that has the least number of active connections. With the least-connected method, the load balancer compares the number of currently active connections to each server, then sends the request to the server with the fewest connections. You set the configuration by using the `least_conn` directive.

- **IP Hash.** This method uses a hash-function to determine which server to select for the next request, which is based on the client’s IP address. You set the configuration by using the `ip_hash` directive.

For more information, see Chapter 4, *Setting Up Load Balancing by Using NGINX*.

See also [https://docs.nginx.com/nginx/](https://docs.nginx.com/nginx/).
Chapter 2 Setting Up Load Balancing by Using HAProxy

Table of Contents

2.1 Installing and Configuring HAProxy ................................................................. 5
2.2 HAProxy Configuration Directives ..................................................................... 5
2.3 Configuring Round Robin Load Balancing by Using HAProxy .............................. 6
2.4 Using Weighted Round Robin Load Balancing with HAProxy .............................. 7
2.5 Adding Session Persistence for HAProxy ......................................................... 8

This chapter describes how to configure load balancing by using HAProxy. The chapter also includes configuration scenarios and examples.

2.1 Installing and Configuring HAProxy

Before you can set up load balancing by using HAProxy, you must first install and configure the feature.

To install HAProxy:

1. Install the haproxy package on each front-end server:

   # dnf install haproxy

2. Edit the /etc/haproxy/haproxy.cfg file to configure HAProxy on each server.

   See Section 2.2, “HAProxy Configuration Directives”.

3. Enable access to the services or ports that you want HAProxy to handle.

   To allow incoming TCP requests on port 80, use the following command:

   # firewall-cmd --zone=zone --add-port=80/tcp
   # firewall-cmd --permanent --zone=zone --add-port=80/tcp

4. Enable and start the haproxy service on each server:

   # systemctl enable --now haproxy

   If you change the HAProxy configuration, reload the haproxy service:

   # systemctl reload haproxy

2.2 HAProxy Configuration Directives

The /etc/haproxy/haproxy.cfg configuration file is divided into the following sections:

**global**

Defines global settings, such as the syslog facility and level to use for logging, the maximum number of concurrent connections that are allowed, and how many processes to start in daemon mode.

**defaults**

Defines the default settings for subsequent sections.

**listen**

Defines a complete proxy, which implicitly includes the frontend and backend components.

**frontend**

Defines the ports that accept client connections.
2.3 Configuring Round Robin Load Balancing by Using HAProxy

The following example uses HAProxy to implement a front-end server that balances incoming requests between two back-end web servers, and which also handles service outages on the back-end servers.

The following figure shows an HAProxy server (10.0.0.10), which is connected to an externally facing network (10.0.0.0/24) and to an internal network (192.168.1.0/24). Two web servers, websrv1 (192.168.1.71) and websrv2 (192.168.1.72), are accessible on the internal network. The IP address 10.0.0.10 is in the private address range 10.0.0.0/24, which cannot be routed on the Internet. An upstream Network Address Translation (NAT) gateway or a proxy server provides access to and from the Internet.

The following is an example configuration in /etc/haproxy/haproxy.cfg on the server:

```
global
daemon
  log 127.0.0.1 local0 debug
  maxconn 50000
  nbproc 1
defaults
  mode http
  timeout connect 5s
  timeout client 25s
  timeout server 25s
  timeout queue 10s

# Handle Incoming HTTP Connection Requests
listen  http-incoming
  mode http
  bind 10.0.0.10:80
# Use each server in turn, according to its weight value
balance roundrobin
# Verify that service is available
  option httpchk HEAD / HTTP/1.1\r\nHost:\ www
```
Using Weighted Round Robin Load Balancing with HAProxy

2.4 Using Weighted Round Robin Load Balancing with HAProxy

HAProxy can also be configured to use the weighted round-robin algorithm to distribute traffic. This algorithm selects servers in turns, according to their weights, and distributes the server load without implementing certain other factors such as server response time. Weighted round-robin allows you to balance traffic proportionally between servers based on processing power and resources available to a server.

To implement weighted round-robin, simply append weight values to each server in your configuration. For example, to distribute twice the amount of traffic to \texttt{websrv1}, change the configuration to include different weight ratios, as follows:

```
server websrv1 192.168.1.71:80 weight 2 maxconn 512 check
server websrv2 192.168.1.72:80 weight 1 maxconn 512 check
```
2.5 Adding Session Persistence for HAProxy

HAProxy provides a multitude of load balancing algorithms, some of which provide features that automatically help to make sure that web sessions have persistent connections to the same back-end server. For instance, you can configure a balance algorithm such as `hdr`, `rdp-cookie`, `source`, `uri`, or `url_param` to ensure that traffic is always routed to the same web server for a particular incoming connection during the session. For example, the `source` algorithm creates a hash of the source IP address and maps it to a particular back-end server. If you use the `rdp-cookie`, or `url_param` algorithms, you may need to configure your back-end web servers or your web applications to facilitate these mechanisms.

If your implementation requires the use of the `leastconn`, `roundrobin`, or `static-rr` algorithm, you can achieve session persistence by using server-dependent cookies.

To enable session persistence for all pages on a web server, use the `cookie` directive to define the name of the cookie to be inserted and add the `cookie` option and server name to the `server` lines, for example:

```plaintext
cookie WEBSVR insert
server websrv1 192.168.1.71:80 weight 1 maxconn 512 cookie 1 check
server websrv2 192.168.1.72:80 weight 1 maxconn 512 cookie 2 check
```

HAProxy includes an additional `Set-Cookie:` header that identifies the web server in its response to the client, for example: `Set-Cookie: WEBSVR=N; path=page_path`. If a client subsequently specifies the `WEBSVR` cookie in a request, HAProxy forwards the request to the web server whose `server` `cookie` value matches the value of `WEBSVR`.

To enable persistence selectively on a web server, use the `cookie` directive to specify that HAProxy should expect the specified cookie, usually a session ID cookie or other existing cookie, to be prefixed with the `server` `cookie` value and a `~` delimiter, for example:

```plaintext
cookie SESSIONID prefix
server websrv1 192.168.1.71:80 weight 1 maxconn 512 cookie 1 check
server websrv2 192.168.1.72:80 weight 1 maxconn 512 cookie 2 check
```

If the value of `SESSIONID` is prefixed with a `server` `cookie` value, for example: `Set-Cookie: SESSIONID=N-Session_ID;`, HAProxy strips the prefix and delimiter from the `SESSIONID` cookie before forwarding the request to the web server whose `server` `cookie` value matches the prefix.

The following example demonstrates how using a prefixed cookie enables session persistence:

```plaintext
$ while true; do curl http://10.0.0.10 cookie "SESSIONID=1~1234;"; sleep 1; done
This is HTTP server websrv1 (192.168.1.71).
This is HTTP server websrv1 (192.168.1.71).
This is HTTP server websrv1 (192.168.1.71).
```

A real web application would usually set the session ID on the server side, in which case the first HAProxy response would include the prefixed cookie in the `Set-Cookie:` header.
Chapter 3 Setting Up Load Balancing by Using Keepalived

Table of Contents

3.1 Installing and Configuring Keepalived ................................................................. 9
3.2 Keepalived Configuration Directives ..................................................................... 9
3.3 Setting Up Load Balancing in NAT Mode by Using Keepalived ................................. 10
    3.3.1 Configuring Firewall Rules for Keepalived NAT-Mode Load Balancing ................. 14
    3.3.2 Configuring Back-End Server Routing for Keepalived NAT-Mode Load Balancing ... 15
3.4 Setting Up Load Balancing by Using Keepalived With HAProxy ............................... 15

This chapter includes tasks and examples that describe how to configure load balancing NAT mode by using Keepalived. The chapter also includes a configuration scenario that demonstrates how to combine the use of Keepalived and HAProxy for high-availability load balancing.

3.1 Installing and Configuring Keepalived

Before you can set up load balancing by using Keepalived, you must install and configure the feature.

1. Install the keepalived package on each server:

   # dnf install keepalived

2. Edit /etc/keepalived/keepalived.conf to configure Keepalived on each server. See Section 3.2, “Keepalived Configuration Directives”.

3. Enable IP forwarding in /etc/sysctl.conf:

   net.ipv4.ip_forward = 1

4. Verify that the IP forwarding has been applied:

   # sysctl -p
   net.ipv4.ip_forward = 1

5. Add firewall rules to allow VRRP communication by using the multicast IP address 224.0.0.18 and the VRRP protocol (112) on each network interface that Keepalived will control, for example:

   # firewall-cmd --direct --permanent --add-rule ipv4 filter INPUT 0 --in-interface enp0s8 --destination 224.0.0.18 --protocol vrrp -j ACCEPT
   # firewall-cmd --direct --permanent --add-rule ipv4 filter OUTPUT 0 --out-interface enp0s8 --destination 224.0.0.18 --protocol vrrp -j ACCEPT
   # firewall-cmd --reload

6. Enable and start the keepalived service on each server:

   # systemctl enable --now keepalived

If you change the Keepalived configuration, reload the keepalived service:

   # systemctl reload keepalived

3.2 Keepalived Configuration Directives

The /etc/keepalived/keepalived.conf configuration file is divided into the following sections:
global_defs
Defines global settings such as the email addresses for sending notification messages, the IP address of an SMTP server, the timeout value for SMTP connections in seconds, a string that identifies the host machine, the VRRP IPv4 and IPv6 multicast addresses, and whether SNMP traps should be enabled.

static_ipaddress, static_routes
Define static IP addresses and routes, which VRRP cannot change. These sections are not required if the addresses and routes are already defined on the servers and these servers already have network connectivity.

vrrp_sync_group
Defines a VRRP synchronization group of VRRP instances that fail over together.

vrrp_instance
Defines a moveable virtual IP address for a member of a VRRP synchronization group's internal or external network interface, which accompanies other group members during a state transition. Each VRRP instance must have a unique value of virtual_router_id, which identifies which interfaces on the master and back-up servers can be assigned a given virtual IP address. You can also specify scripts that are run on state transitions to BACKUP, MASTER, and FAULT, and whether to trigger SMTP alerts for state transitions.

vrrp_script
Defines a tracking script that Keepalived can run at regular intervals to perform monitoring actions from a vrrp_instance or vrrp_sync_group section.

virtual_server_group
Defines a virtual server group, which allows a real server to be a member of several virtual server groups.

virtual_server
Defines a virtual server for load balancing, which is composed of several real servers.

For more information about setting up load balancing with Keepalived, see Chapter 3, Setting Up Load Balancing by Using Keepalived

3.3 Setting Up Load Balancing in NAT Mode by Using Keepalived

The following example shows how you would use Keepalived in NAT mode to implement a basic failover and load balancing configuration on two servers. One server acts as the master, the other acts as a backup, with the master server having a higher priority than the back-up server. They use VRRP to monitor the current routing state, and you can read more about how that works and why it is useful in Using Keepalived With VRRP.

Each of the servers has two network interfaces, where one interface is connected to the side-facing an external network (192.168.1.0/24), and the other interface is connected to an internal network (10.0.0.0/24), on which two web servers are accessible.

The following figure shows that the Keepalived master server has the following network addresses: 192.168.1.10, 192.168.1.1 (virtual), 10.0.0.10, and 10.0.0.100 (virtual).

The Keepalived back-up server has he following network addresses: 192.168.1.11 and 10.0.0.11.

The web servers webserv1 and webserv2 have the network addresses 10.0.0.71 and 10.0.0.72, respectively.
The following is an example of the configuration in the `/etc/keepalived/keepalived.conf` file on the master server:

```conf

global_defs {
    notification_email {
        root@example.com
    }
    notification_email_from srv1@example.com
    smtp_server localhost
    smtp_connect_timeout 30
}

vrrp_sync_group VRRP1 {
    # Group the external and internal VRRP instances so they fail over together
    group {
        external
        internal
    }
}

vrrp_instance external {
    state MASTER
    interface enp0s8
    virtual_router_id 91
    priority 200
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1215
    }
    # Define the virtual IP address for the external network interface
    virtual_ipaddress {
        192.168.1.1/24
    }
}

vrrp_instance internal {
    state MASTER
    interface enp0s9
    virtual_router_id 92
}
```

**Figure 3.1 Keepalived Configuration for Load Balancing in NAT Mode**
Setting Up Load Balancing in NAT Mode by Using Keepalived

priority 200
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1215
}
# Define the virtual IP address for the internal network interface
virtual_ipaddress {
    10.0.0.100/24
}

# Define a virtual HTTP server on the virtual IP address 192.168.1.1
virtual_server 192.168.1.1 80 {
    delay_loop 10
    protocol TCP
    # Use round-robin scheduling in this example
    lb_algo rr
    # Use NAT to hide the back-end servers
    lb_kind NAT
    # Persistence of client sessions times out after 2 hours
    persistence_timeout 7200
    real_server 10.0.0.71 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 5
            connect_port 80
        }
    }
    real_server 10.0.0.72 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 5
            connect_port 80
        }
    }
}

The previous configuration includes an additional vrrp_sync_group section so that the network interfaces are assigned together on failover, as well as a virtual_server section to define the real back-end servers that Keepalived uses for load balancing. The value of lb_kind is set to use NAT, which means the Keepalived server handles both inbound and outbound network traffic from and to the client on behalf of the back-end servers.

The configuration of the back-up server is the same, except for the values of notification_email_from, state, priority, and possibly interface, if the system hardware configuration is different:

global_defs {
    notification_email {
        root@example.com
    }
    notification_email_from srv2@example.com
    smtp_server localhost
    smtp_connect_timeout 30
}
vrrp_sync_group VRRP1 {
    # Group the external and internal VRRP instances so they fail over together
    group {
        external
        internal
    }
}
Setting Up Load Balancing in NAT Mode by Using Keepalived

vrrp_instance external {
  state BACKUP
  interface enp0s8
  virtual_router_id 91
  priority 100
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass 1215
  }
  # Define the virtual IP address for the external network interface
  virtual_ipaddress {
    192.168.1.1/24
  }
}

vrrp_instance internal {
  state BACKUP
  interface enp0s9
  virtual_router_id 92
  priority 100
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass 1215
  }
  # Define the virtual IP address for the internal network interface
  virtual_ipaddress {
    10.0.0.100/24
  }
}

# Define a virtual HTTP server on the virtual IP address 192.168.1.1
virtual_server 192.168.1.1 80 {
  delay_loop 10
  protocol TCP
  # Use round-robin scheduling in this example
  lb_algo rr
  # Use NAT to hide the back-end servers
  lb_kind NAT
  # Persistence of client sessions times out after 2 hours
  persistence_timeout 7200
  real_server 10.0.0.71 80 {
    weight 1
    TCP_CHECK {
      connect_timeout 5
      connect_port 80
    }
  }
  real_server 10.0.0.72 80 {
    weight 1
    TCP_CHECK {
      connect_timeout 5
      connect_port 80
    }
  }
}

The following additional configuration changes are required:

- Configure the firewall rules on each Keepalived server (master and backup) that you are configuring as a load balancer. See Section 3.3.1, “Configuring Firewall Rules for Keepalived NAT-Mode Load Balancing”.
• Configure a default route for the virtual IP address of the load balancer's internal network interface on each back-end server that you intend to use with the Keepalived load balancer. See Section 3.3.2, “Configuring Back-End Server Routing for Keepalived NAT-Mode Load Balancing”.

See Section 3.1, “Installing and Configuring Keepalived” for more information.

3.3.1 Configuring Firewall Rules for Keepalived NAT-Mode Load Balancing

If you configure Keepalived to use NAT mode for load balancing with the servers on the internal network, the Keepalived server handles all inbound and outbound network traffic and hides the existing back-end servers by rewriting the source IP address of the real back-end server in outgoing packets with the virtual IP address of the external network interface.

To configure a Keepalived server to use NAT mode for load balancing:

1. Configure the firewall so that the interfaces on the external network side are in a zone that is different from the interfaces on the internal network side.

The following example demonstrates how to move interface enp0s9 to the internal zone, while interface enp0s8 remains in the public zone:

```bash
# firewall-cmd --get-active-zones
public
    interfaces: enp0s8 enp0s9
# firewall-cmd --zone=public --remove-interface=enp0s9
# firewall-cmd --zone=internal --add-interface=enp0s9
# firewall-cmd --zone=public --remove-interface=enp0s9
# firewall-cmd --zone=internal --add-interface=enp0s9
# firewall-cmd --get-active-zones
internal
    interfaces: enp0s9
public
    interfaces: enp0s8
```

2. Configure NAT mode (masquerading) on the external network interface, for example:

```bash
# firewall-cmd --zone=public --add-masquerade
# firewall-cmd --permanent --zone=public --add-masquerade
# firewall-cmd --zone=public --query-masquerade
yes
# firewall-cmd --zone=internal --query-masquerade
no
```

3. If not already enabled for your firewall, configure forwarding rules between the external and internal network interfaces, for example:

```bash
# firewall-cmd --direct --permanent --add-rule ipv4 filter FORWARD 0 \ -i enp0s8 -o enp0s9 -m state --state RELATED,ESTABLISHED -j ACCEPT
# firewall-cmd --direct --permanent --add-rule ipv4 filter FORWARD 0 \ -i enp0s9 -o enp0s8 -j ACCEPT
# firewall-cmd --direct --permanent --add-rule ipv4 filter FORWARD 0 \ -j REJECT --reject-with icmp-host-prohibited
# firewall-cmd --reload
```

4. Enable access to the services or ports that you want Keepalived to handle.
3.3.2 Configuring Back-End Server Routing for Keepalived NAT-Mode Load Balancing

On each back-end real servers that you intend to use with the Keepalived load balancer, ensure that the routing table contains a default route for the virtual IP address of the load balancer's internal network interface.

For example, if the virtual IP address is 10.0.0.100, use the `ip` command to examine the routing table and to set the default route:

```bash
# ip route show
10.0.0.0/24 dev enp0s8 proto kernel scope link src 10.0.0.71
# ip route add default via 10.0.0.100 dev enp0s8
# ip route show
default via 10.0.0.100 dev enp0s8
10.0.0.0/24 dev enp0s8 proto kernel scope link src 10.0.0.71
```

To make the default route for `enp0s8` persist across system reboots, create the `/etc/sysconfig/network-scripts/route-enp0s8` file:

```bash
# echo "default via 10.0.0.100 dev enp0s8" > /etc/sysconfig/network-scripts/route-enp0s8
```

3.4 Setting Up Load Balancing by Using Keepalived With HAProxy

You can use Keepalived to provide failover services for back-up routers, while at the same time also using HAProxy for load balancing and to achieve high availability across distributed servers. The advantage of this approach is that the packet and application layers are separated, which means the health checks that are performed by Keepalived for the load-balancing servers are not impacted by the inbound HTTP or TCP traffic that HAProxy is managing. Also, failover routing, which is achieved by using VRRP, can be activated automatically without waiting for a client response to time out. To learn more about the usefulness of VRRP, see Using Keepalived With VRRP.

The practicality of using this method is that if your public-facing HAProxy load balancer goes offline, Keepalived automatically detects that the load balancer has gone offline and dynamically switches to another HAProxy server. If the Keepalived master router goes offline, you can safely rely on the VRRP settings that you configured to ensure that traffic is automatically handled by your Keepalived back-up router.

The role of HAProxy in the setup is to provide inbound load balancing and session persistence to your back-end servers: Keepalived is solely responsible for monitoring the status of HAProxy and providing an alternative routing mechanism. Using both tools in combination provides a highly-available and resilient load-balancing solution.

The following example provides instructions that are similar to the instructions in Section 3.3, "Setting Up Load Balancing in NAT Mode by Using Keepalived". However, in the following example, HAProxy is installed on both the Keepalived master server, as well as the Keepalived back-up server.

Like the other example, the external virtual IP address is 192.168.1.1, which is on the 192.168.1.0/24 external network. This IP address is dynamically assigned through NAT (between the Keepalived master server), with the IP address 192.168.1.10 (and the Keepalived back-up server), with the external IP address 192.168.1.11.

The internal network is hosted on the 10.0.0.0/24 subnet, with websvr1 and websvr2 assigned the 10.0.0.71 and 10.0.0.72 IP addresses, respectively.
The following example shows the configuration in the `/etc/keepalived/keepalived.conf` file on the master server:

```conf

global_defs {
    notification_email {
        root@example.com
    }
    notification_email_from srv1@example.com
    smtp_server localhost
    smtp_connect_timeout 30
}

vrrp_sync_group vg1 {
    group {
        external
        internal
    }
}

vrrp_script chk_haproxy {
    script "killall -0 haproxy" # check the haproxy process
    interval 2 # every 2 seconds
    weight 2 # add 2 points if OK
}

vrrp_instance external {
    state MASTER
    interface enp0s8
    virtual_router_id 91
    priority 200
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1215
    }
    virtual_ipaddress {
```
192.168.1.1/24
}
track_script {
chk_haproxy
}
}

vrrp_instance internal {
  state MASTER
  interface enp0s9
  virtual_router_id 92
  priority 200
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass 1215
  }
  virtual_ipaddress {
    10.0.0.100/24
  }
}

In the previous example, the configuration for the back-up Keepalived server is identical, but the `state` value must be set to `BACKUP`. You do not need to set up a `virtual_server`, because in this scenario, Keepalived is only used to route traffic, not to perform load balancing.

For more information about configuring Keepalived and setting the appropriate firewall rules, see Chapter 3, Setting Up Load Balancing by Using Keepalived.

The HAProxy settings are configured in the `/etc/haproxy/haproxy.cfg` file. For both HAProxy installations they should be identical, as Keepalived dynamically routes from one configuration to the other automatically, as needed:

```
global
daemon
  log 127.0.0.1 local0 debug
maxconn 4000
nbproc 1

defaults
  mode        http
retries      3
timeout connect 5s
timeout client 25s
timeout server 25s
timeout queue 10s

listen http-incoming
  mode http
  bind internal-server-ip:80
  option http-server-close
  option forwardfor
default_backend app

backend app
  balance roundrobin
  option httpchk HEAD / HTTP/1.1\r\nHost: localhost
  option httpclose
  option forwardfor
  server websrv1 192.168.1.71:80 weight 1 maxconn 512 check
  server websrv2 192.168.1.72:80 weight 1 maxconn 512 check
```

In the previous example the `option http-server-close` and `option httpclose` options are used to terminate idle connections. This configuration demonstrates the round-robin, load-balancing strategy.
If no option is specified, then HAProxy defaults to using the `option http-keep-alive` option, which keeps any new connections open until every request and response journey that is associated with them is processed.

For more information about configuring HAProxy and setting the appropriate firewall rules, see Chapter 2, *Setting Up Load Balancing by Using HAProxy*. 
Chapter 4 Setting Up Load Balancing by Using NGINX

Table of Contents

4.1 Installing NGINX ............................................................... 19
4.2 NGINX Configuration Directives ....................................... 19
4.3 Configuring Round Robin Load Balancing by Using NGINX ............................................. 19
4.4 Using Weighted Round Robin Load Balancing With NGINX ........................................... 21
4.5 Using Least-Connected Load Balancing With NGINX ..................................................... 22
4.6 Adding Session Persistence for NGINX .................................. 23

This chapter describes how to configure NGINX as a load balancer and includes installation instructions and configuration directives. For an overview of NGINX, see Section 1.5, “About NGINX”.

4.1 Installing NGINX

Before you can use NGINX for load balancing, you must first install the software and configure the environment.

1. Install the nginx package on each server:

```
# dnf install nginx
```

Depending on your intended configuration, you may need to install additional modules. The nginx-all-modules metapackage installs all of the packages. To see a listing of all of the available modules in the package manager, use the following command:

```
# dnf search nginx-mod*
```

Note that if you intend to do TCP/UDP load balancing, you must install the nginx-mod-stream module package.

2. Enable access to the services or ports that you want NGINX to handle.

For example, you would allow incoming TCP requests on port 80 as follows:

```
# firewall-cmd --zone=zone --add-port=80/tcp
# firewall-cmd --permanent --zone=zone --add-port=80/tcp
```

3. If SELinux is set to enforcing mode on the system, add a rule to allow NGINX to relay HTTP traffic to any configured back-end servers:

```
# setsebool httpd_can_network_relay on
```

4. Enable and start the nginx service on the server:

```
# systemctl enable --now nginx
```

If you change the NGINX configuration, reload the nginx service:

```
# systemctl reload nginx
```

4.2 NGINX Configuration Directives

NGINX configuration can be spread across several files to specify different configuration directives and set the values for configuration variables. Configuration is stored in /etc/nginx. The base configuration is stored in /etc/nginx/nginx.conf, while site specific configuration tends to be created within distinct
files in /etc/nginx/conf.d/. By convention, site configurations tend to use the full domain name for the file name and should have a .conf suffix.

In these examples, a configuration has the following general format:

```plaintext
http {
    server {
        listen 80;
        listen [:]:80;
        server_name example.com www.example.com;
        location / {
            root /usr/share/nginx/html/example.com;
            index index.html;
        }
    }
}
```

The previous example shows an HTTP server configuration for a web server that serves content from the web root directory at /usr/share/nginx/html/example.com.

The following configuration directives are useful to note for the purpose of configuring load balancing:

- **http, https, stream**
  - Defines the protocol for which the settings apply. Use https for TLS connections to the load balancer and stream for generic TCP/UDP traffic.

- **server**
  - Defines how to handle incoming traffic from the specified ports for the chosen protocol.

  To configure at least one listening port for IPv4, use the listen keyword:

  ```plaintext
  listen 80;
  ```

  To listen on IPv6 interfaces, prepend the [:] directive to the port number, for example:

  ```plaintext
  listen [:]:80
  ```

  Note that the listen lines can be duplicated to specify more than one port for a server{} block.

- **server_name**
  - Use the server_name keyword to define the hostname or domain name that the server responds to. If you do not specify this value, the configuration applies to any incoming connection, however you may need to comment out the default server configuration within /etc/nginx/nginx.conf to avoid conflicting configuration definitions.

- **location**
  - The location directive defines path mappings and behavior, depending on incoming requests on the server. At minimum, you must have a value for the web root that is indicated with the value /. The behavior is defined by setting values within a location block.

  For example, to configure a simple web server that serves content from a directory on the server, use the root keyword and specify the directory that the content is located.

  ```plaintext
  root /usr/share/nginx/html/example.com;
  index index.html;
  ```

  The proxy_pass directive can be used to implement a reverse proxy service. Traffic is proxied onto the specified server or group of servers, as defined in an upstream directive.
For example, you would proxy inbound HTTP traffic to a website that is hosted on websr1.example.com on port 9090 as follows:

```
server {
    location / {
        proxy_pass http://websr1.example.com:9090
    }
}
```

You can also specify a server group by referencing its defined `upstream` name.

**upstream**

An `upstream` directive is used to define a group of one or more servers where the content is located and which can be used by the `proxy_pass` directive. For example, you can create an upstream group of servers called `backend` as follows:

```
upstream backend {
    server server1.example.com;
    server server2.example.com;
    server server3.example.com;
}
```

To use this group, the `proxy_pass` directive is specified:

```
proxy_pass http://backend
```

The `upstream` directive is the key configuration component that is used to control load-balancing methods and algorithms. For more information, see [http://nginx.org/en/docs/http/ngx_http_upstream_module.html](http://nginx.org/en/docs/http/ngx_http_upstream_module.html).

### 4.3 Configuring Round Robin Load Balancing by Using NGINX

The default load balancing method that is used by NGINX is the round-robin method. This method proxies traffic sequentially to each server in a defined group.

Create a configuration file for the load-balancer at `/etc/nginx/conf.d/example.com.conf`, where `example.com` is the name of the external domain where inbound traffic is directed. The file should contain the following content:

```
http
{
    upstream backend {
        server server1.example.com;
        server server2.example.com;
        server server3.example.com;
    }
    server {
        listen 80;
        server_name example.com www.example.com;
        location / {
            proxy_pass http://backend;
        }
    }
}
```

In the `upstream backend` configuration block, list the back-end servers within your environment. For example, you should substitute `server1.example.com` with the fully qualified domain name or the hostname of a web server instance.
Set the `server_name` directive with the domain name or names that you intend to use publicly for the load balanced service. For example, substitute `example.com` and `www.example.com` to match your own domain.

You can optionally append additional failover options, such as `max_fails` and `fail_timeout`, to the end of each entry to add resilience in the event that any of the servers goes offline.

After ensuring that the configuration is valid, enable it by reloading NGINX on the public-facing and back-end servers:

```
# systemctl reload nginx
```

### 4.4 Using Weighted Round Robin Load Balancing With NGINX

When using servers with varying physical locations or differing hardware resources, you can configure NGINX to allocate more of the traffic to servers that provide less latency and can handle more of a load. This method is referred to as the weighted round-robin method.

Weighted round-robin configuration is accomplished by appending a `weight` value to the end of each entry in the server group section of the NGINX site configuration file. Set the weight of your slowest server to 1, and then set the weight of other servers relative to that setting.

The following example demonstrates how servers can handle multiple times the load of the base server. One server receives twice the amount of traffic, while the other server receives four times the amount:

```
upstream backend {
  server server1.example.com weight=1;
  server server2.example.com weight=2;
  server server3.example.com weight=4;
}
```

Reload NGINX to apply the new configuration:

```
# systemctl reload nginx
```

### 4.5 Using Least-Connected Load Balancing With NGINX

The least-connected load balancing method is used to automatically control the load on application instances, mostly in situations where different inbound requests might take longer to process than other requests.

If you are using the least-connected load balancing method, NGINX always directs new incoming requests to the server with the least number of currently active requests. This load balancing strategy is intended to ensure that no busy servers are overloaded with new requests, while other servers that are capable of taking on the load remain idle.

You can activate the least-connected load balancing method for NGINX by specifying the `least-conn` directive as part of the server group configuration, for example:

```
upstream backend {
  least_conn;
  server server1.example.com;
  server server2.example.com;
  server server3.example.com;
}
```

Reload NGINX to apply the new configuration:

```
# systemctl reload nginx
```
4.6 Adding Session Persistence for NGINX

If you are performing load balancing of a web application, it can be useful to ensure that the same back-end server that handled inbound requests continues to do so for the same source. It is particularly important when a website or web service must preserve log-in sessions between requests, cancel an existing request, or monitor the progress of large back-end transactions.

To achieve this behavior, activate the IP hash method for NGINX by specifying the `ip_hash` directive as part of the server group configuration, for example:

```plaintext
upstream backend {
    ip_hash;
    server server1.example.com;
    server server2.example.com;
    server server3.example.com;
}
```

Reload NGINX to apply the new configuration:

```plaintext
# systemctl reload nginx
```