Red Hat Enterprise Linux 7 High Availability Add-On Overview

Overview of the High Availability Add-On for Red Hat Enterprise Linux 7

Red Hat Engineering Content Services

Red Hat Enterprise Linux 7 High Availability Add-On Overview

Overview of the High Availability Add-On for Red Hat Enterprise Linux 7

Red Hat Engineering Content Services
docs-need-a-fix@redhat.com

Abstract

Red Hat High Availability Add-On Overview provides an overview of the High Availability Add-On for Red Hat Enterprise Linux 7.

Table of Contents

Chapter 1. High Availability Add-On Overview

The High Availability Add-On is a clustered system that provides reliability, scalability, and availability to critical production services. The following sections provide a high-level description of the components and functions of the High Availability Add-On:

- Section 1.1, “Cluster Basics”
- Section 1.2, “High Availability Add-On Introduction”
- Section 1.4, “Pacemaker Architecture Components”

1.1. Cluster Basics

A cluster is two or more computers (called *nodes* or *members*) that work together to perform a task. There are four major types of clusters:

- Storage
- High availability
- Load balancing
- High performance

Storage clusters provide a consistent file system image across servers in a cluster, allowing the servers to simultaneously read and write to a single shared file system. A storage cluster simplifies storage administration by limiting the installation and patching of applications to one file system. Also, with a cluster-wide file system, a storage cluster eliminates the need for redundant copies of application data and simplifies backup and disaster recovery. The High Availability Add-On provides storage clustering in conjunction with Red Hat GFS2 (part of the Resilient Storage Add-On).

high availability clusters provide highly available services by eliminating single points of failure and by failing over services from one cluster node to another in case a node becomes inoperative. Typically, services in a high availability cluster read and write data (via read-write mounted file systems). Therefore, a high availability cluster must maintain data integrity as one cluster node takes over control of a service from another cluster node. Node failures in a high availability cluster are not visible from clients outside the cluster. (high availability clusters are sometimes referred to as failover clusters.) The High Availability Add-On provides high availability clustering through its High Availability Service Management component, **`Pacemaker`**.

Load-balancing clusters dispatch network service requests to multiple cluster nodes to balance the request load among the cluster nodes. Load balancing provides cost-effective scalability because you can match the number of nodes according to load requirements. If a node in a load-balancing cluster becomes inoperative, the load-balancing software detects the failure and redirects requests to other cluster nodes. Node failures in a load-balancing cluster are not visible from clients outside the cluster. Load balancing is available with the Load Balancer Add-On.

High-performance clusters use cluster nodes to perform concurrent calculations. A high-performance cluster allows applications to work in parallel, therefore enhancing the performance of the applications. (High performance clusters are also referred to as computational clusters or grid computing.)

Note

The cluster types summarized in the preceding text reflect basic configurations; your needs might require a combination of the clusters described.

Additionally, the Red Hat Enterprise Linux High Availability Add-On contains support for configuring and managing high availability servers *only*. It *does not* support high-performance clusters.

1.2. High Availability Add-On Introduction

The High Availability Add-On is an integrated set of software components that can be deployed in a variety of configurations to suit your needs for performance, high availability, load balancing, scalability, file sharing, and economy.

The High Availability Add-On consists of the following major components:

- Cluster infrastructure — Provides fundamental functions for nodes to work together as a cluster: configuration-file management, membership management, lock management, and fencing.
- High availability Service Management — Provides failover of services from one cluster node to another in case a node becomes inoperative.
- Cluster administration tools — Configuration and management tools for setting up, configuring, and managing a the High Availability Add-On. The tools are for use with the Cluster Infrastructure components, the high availability and Service Management components, and storage.

You can supplement the High Availability Add-On with the following components:

- Red Hat GFS2 (Global File System 2) — Part of the Resilient Storage Add-On, this provides a cluster file system for use with the High Availability Add-On. GFS2 allows multiple nodes to share storage at a block level as if the storage were connected locally to each cluster node. GFS2 cluster file system requires a cluster infrastructure.
- Cluster Logical Volume Manager (CLVM) — Part of the Resilient Storage Add-On, this provides volume management of cluster storage. CLVM support also requires cluster infrastructure.
- Load Balancer Add-On — Routing software that provides high availability load balancing and failover in layer 4 (TCP) and layer 7 (HTTP, HTTPS) services. the Load Balancer Add-On runs in a cluster of redundant virtual routers that uses load algorhithms to distribute client requests to real servers, collectively acting as a virtual server.

1.3. Pacemaker Overview

The High Availability Add-On cluster infrastructure provides the basic functions for a group of computers (called *nodes* or *members*) to work together as a cluster. Once a cluster is formed using the cluster infrastructure, you can use other components to suit your clustering needs (for example, setting up a cluster for sharing files on a GFS2 file system or setting up service failover). The cluster infrastructure performs the following functions:

- Cluster management
- Lock management
- Fencing

- Cluster configuration management

1.4. Pacemaker Architecture Components

A cluster configured with Pacemaker comprises separate component daemons that monitor cluster membership, scripts that manage the services, and resource management subsystems that monitor the disparate resources. The following components form the Pacemaker architecture:

Cluster Information Base (CIB)

The Pacemaker information daemon, which uses XML internally to distribute and synchronize current configuration and status information from the Designated Co-ordinator (DC) — a node assigned by Pacemaker to store and distribute cluster state and actions via CIB — to all other cluster nodes.

Cluster Resource Management Daemon (CRMd)

Pacemaker cluster resource actions are routed through this daemon. Resources managed by CRMd can be queried by client systems, moved, instantiated, and changed when needed.

Each cluster node also includes a local resource manager daemon (LRMd) that acts as an interface between CRMd and resources. LRMd passes commands from CRMd to agents, such as starting and stopping and relaying status information.

Shoot the Other Node in the Head (STONITH)

Often deployed in conjunction with a power switch, STONITH acts as a cluster resource in Pacemaker that processes fence requests, forcefully powering down nodes and removing them from the cluster to ensure data integrity. STONITH is confugred in CIB) and can be monitored as a normal cluster resource.

1.5. Pacemaker Configuration and Management Tools

Pacemaker features two configuration tools for cluster deployment, monitoring, and management.

pcs

pcs can control all aspects of Pacemaker and the Corosync heartbeat daemon. A command-line based program, **pcs** can perform the following cluser management tasks:

- Create and configure a Pacemaker/Corosync cluster
- Modify configuration of the cluster while it is running
- Remotely configure both Pacemaker and Corosync remotely as well as start, stop, and display status information of the cluster.

pcsd

A Web-based graphical user interface to create and configure Pacemaker/Corosync clusters, with the same features and abilities as the command-line based **pcs** utility.

Chapter 2. Cluster Operation

This chapter provides a summary of the various cluster functions and features. From establishing cluster quorum to node fencing for isolation, these disparate features comprise the core functionality of the High Availability Add-On.

2.1. Quorum Overview

In order to maintain cluster integrity and availability, cluster systems use a concept known as *quorum* to prevent data corruption and loss. A cluster has quorum when more than half of the cluster nodes are online. To mitigate the chance of of data corruption due to failure, Pacemaker by default stops all resources if the cluster does not have quorum.

Quorum is established using a voting system. When a cluster node does not function as it should or loses communication with the rest of the cluster, the majority working nodes can vote to isolate and, if needed, fence the node for servicing.

For example, in a 6-node cluster, quorum is established when at least 4 cluster nodes are functioning. If the majority of nodes go offline or become unavailable, the cluster no longer has quorum Pacemaker stops clustered services.

The quorum features in Pacemaker prevent what is also known as *split-brain*, a phenomenon where the cluster is separated from communication but each part continues working as separate clusters, potentially writing to the same data and possibly causing corruption or loss.

Quorum support in the High Availability Add-On are provided by a Corosync plugin called **`votequorum`**, which allows administrators to configure a cluster with a number of votes assigned to each system in the cluster and ensuring that only when a majority of the votes are present, cluster operations are allowed to proceed.

In a situation where there is no majority (such as an odd-numbered cluster where one node becomes unavailable, resulting in a 50% cluster split), **`votequorum`** can be configured to have a *tiebreaker* policy, which administrators can configure to continue quorum using the remaining cluster nodes that are still in contact with the available cluster node that has the lowest node ID.

2.2. Fencing Overview

In a cluster system, there can be many nodes working on several pieces of vital production data. Nodes in a busy, multi-node cluster could begin to act erratically or become unavailable, prompting action by administrators. The problems caused by errant cluster nodes can be mitigated by establishing a *fencing* policy.

Fencing is the disconnection of a node from the cluster's shared storage. Fencing cuts off I/O from shared storage, thus ensuring data integrity. The cluster infrastructure performs fencing through the *STONITH* facility.

When Pacemaker determines that a node has failed, it communicates to other cluster-infrastructure components that the node has failed. STONITH fences the failed node when notified of the failure. Other cluster-infrastructure components determine what actions to take, which includes performing any recovery that needs to done. For example, DLM and GFS2, when notified of a node failure, suspend activity until they detect that STONITH has completed fencing the failed node. Upon confirmation that the failed node is fenced, DLM and GFS2 perform recovery. DLM releases locks of the failed node; GFS2 recovers the journal of the failed node.

Node-level fencing via STONITH can be configured with a variety of supported fence devices, including:

- Uninterruptible Power Supply (UPS) — a device containing a battery that can be used to fence devices in event of a power failure
- Power Distribution Unit (PDU) — a device with multiple power outlets used in data centers for clean power distribution as well as fencing and power isolation services
- Blade power control devices — dedicated systems installed in a data center configured to fence cluster nodes in the event of failure
- Lights-out devices — Network-connected devices that manage cluster node availability and can perform fencing, power on/off, and other services by administrators locally or remotely

Chapter 3. Red Hat High Availability Add-On Resources

This chapter provides

3.1. Red Hat High Availability Add-On Resource Overview

A *cluster resource* is an instance of program, data, or application to be managed by the cluster service. These resources are abstracted by *agents* that provide a standard interface for managing the resource in a cluster environment. This standardization is based on industry approved frameworks and classes, which makes managing the availability of various cluster resources transparent to the cluster service itself.

3.2. Red Hat High Availability Add-On Resource Classes

There are several classes of resource agents supported by Red Hat High Availability Add-On:

- LSB — The Linux Standards Base agent abstracts the compliant services supported by the LSB, namely those services in **`/etc/init.d`** and the associated return codes for successful and failed service states (started, stopped, running status).
- OCF — The Open Cluster Framework is superset of the LSB (Linux Standards Base) that sets standards for the creation and execution of server initialization scripts, input parameters for the scripts using environment variables, and more.
- Systemd — The newest system services manager for Linux based systems, Systemd uses sets of unit files rather than initialization scripts as does LSB and OCF. These units can be manually created by administrators or can even be created and managed by services themselves. Pacemaker manages these units in a similar way that it manages OCF or LSB init scripts.
- Upstart — Much like systemd, Upstart is an alternative system initialization manager for Linux. Upstart uses jobs, as opposed to units in systemd or init scripts.
- STONITH — A resource agent exlcusively for fencing services and fence agents using STONITH.
- Nagios — Agents that abstract plugins for the Nagios system and infrastructure monitoring tool.

3.3. Monitoring Resources

To ensure that resources remain healthy, you can add a monitoring operation to a resource's definition. If you do not specify a monitoring operation for a resource, by default the pcs command will create a monitoring operation, with an interval that is determined by the resource agent. If the resource agent does not provide a default monitoring interval, the pcs command will create a monitoring operation with an interval of 60 seconds.

3.4. Resource Constraints

You can determine the behavior of a resource in a cluster by configuring *constraints*. You can configure the following categories of constraints:

- location constraints — A location constraint determines which nodes a resource can run on.
- order constraints — An order constraint determines the order in which the resources run.
- colocation constraints — A colocation constraint determines where resources will be placed relative to other resources.

As a shorthand for configuring a set of constraints that will locate a set of resources together and ensure that the resources start sequentially and stop in reverse order, Pacemaker supports the concept of resource groups.

3.5. Resource Groups

One of the most common elements of a cluster is a set of resources that need to be located together, start sequentially, and stop in the reverse order. To simplify this configuration, Pacemaker supports the concept of *groups*.

You create a resource group with the **`pcs resource`** command, specifying the resources to include in the group. If the group does not exist, this command creates the group. If the group exists, this command adds additional resources to the group. The resources will start in the order you specify them with this command, and will stop in the reverse order of their starting order.

Chapter 4. Load Balancer Overview

The Load Balancer is a set of integrated software components that provide for balancing IP load across a set of real servers. Keepalived uses LVS to perform load balancing and failover tasks on, while HAProxy performs load balancing and high-availability services to TCP and HTTP applications.

Keepalived runs on an *active LVS router* as well as one or more *backup LVS routers*. The active LVS router serves two roles:

- To balance the load across the real servers.
- To check the integrity of the services on each real server.

The active (master) router informs the backup routers of its active status using the *Virtual Router Redundancy Protocol* (VRRP), which requires the master router to send out advertisements at regular intervals. If the active router stops sending advertisements, a new master is elected.

This chapter provides an overview of The Load Balancer Add-On components and functions, and consists of the following sections:

4.1. A Basic Load Balancer Configuration

Figure 4.1, “A Basic Load Balancer Configuration” shows a simple Load Balancer configuration consisting of two layers. On the first layer is one active and one backup LVS router, though having multiple backup routers is supported. Each LVS router has two network interfaces, one interface on the Internet and one on the private network, enabling them to regulate traffic between the two networks. For this example the active router is using *Network Address Translation* or *NAT* to direct traffic from the Internet to a variable number of real servers on the second layer, which in turn provide the necessary services. Therefore, the real servers in this example are connected to a dedicated private network segment and pass all public traffic back and forth through the active LVS router. To the outside world, the servers appear as one entity.

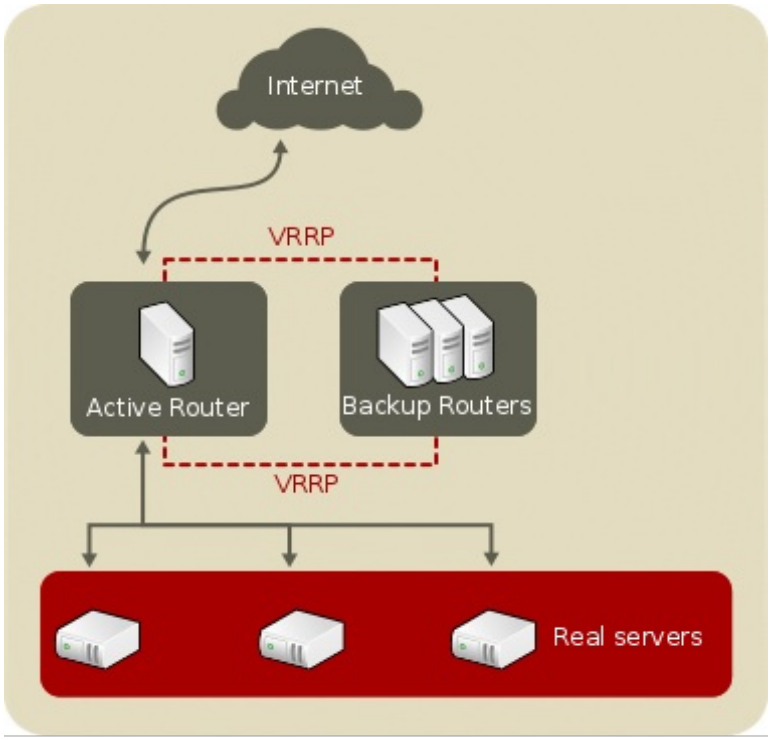

Figure 4.1. A Basic Load Balancer Configuration

Service requests arriving at the LVS router are addressed to a *virtual IP* address, or *VIP*. This is a publicly-routable address the administrator of the site associates with a fully-qualified domain name, such as www.example.com, and is assigned to one or more *virtual servers*. A virtual server is a service configured to listen on a specific virtual IP address and port. A VIP address migrates from one LVS router to the other during a failover, thus maintaining a presence at that IP address (also known as *floating IP addresses*).

VIP addresses may be assigned to the same device which connects the LVS router to the Internet. For instance, if **eth0** is connected to the Internet, then multiple virtual servers can be assigned to **eth0**. Alternatively, each virtual server can be associated with a separate device per service. For example, HTTP traffic can be handled on **eth0** at 192.168.1.111 while FTP traffic can be handled on **eth0** at 192.168.1.222.

Further, administrators can additionally assign services to the same VIP address but on differing ports. So, in the aforementioned example, an administrator can us 192.168.1.111 as a VIP, with port 21 for FTP service and port 80 for HTTP service.

In a deployment scenario involving both one active and one passive router, the role of the active router is to redirect service requests from virtual IP addresses to the real servers. The redirection is based on one of eight supported load-balancing algorithms described further in Section 4.4, “Load Balancer Scheduling Overview”.

The active router also dynamically monitors the overall health of the specific services on the real servers through three built-in health checks: simple TCP connect, HTTP, and HTTPS. For TCP connect, the active router will periodically check that it can connect to the real servers on a certain port. For HTTP and HTTPS, the active router will periodically fetch a URL on the real servers and verify its content.

The backup routers perform the role of standby systems. Router failover is handled by VRRP. On startup, all routers will join a multicast group. This multicast group is used to send and receive VRRP advertisements. Since VRRP is a priority based protocol, the router with the highest priority is elected the master. Once a router has been elected master, it is responsible for sending VRRP advertisements at periodic intervals to the multicast group.

If the backup routers fail to receive advertisements within a certain time period (based on the advertisement interval), a new master will be elected. The new master will take over the VIP and send a gratuitous *Address Resolution Protocol* (ARP) message. When a router returns to active service, it may either become a backup or a master. The behavior is determined by the router's priority.

The simple, two-layered configuration used in Figure 4.1, “A Basic Load Balancer Configuration” is best for serving data which does not change very frequently — such as static webpages — because the individual real servers do not automatically sync data between each node.

4.2. A Three-Tier Load Balancer Configuration

Figure 4.2, “A Three-Tier Load Balancer Configuration” shows a typical three-tier Load Balancer Add-On topology. In this example, the active LVS router routes the requests from the Internet to the pool of real servers. Each of the real servers then accesses a shared data source over the network.

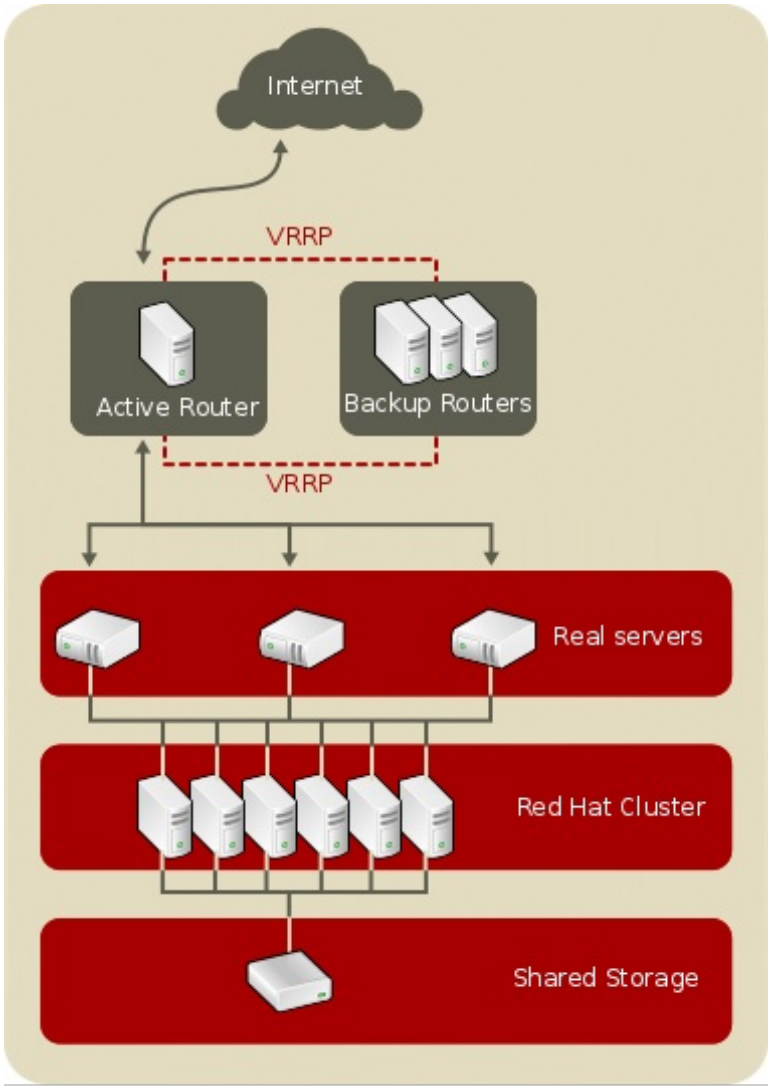

Figure 4.2. A Three-Tier Load Balancer Configuration

This configuration is ideal for busy FTP servers, where accessible data is stored on a central, highly available server and accessed by each real server via an exported NFS directory or Samba share. This topology is also recommended for websites that access a central, highly available database for transactions. Additionally, using an active-active configuration with the Load Balancer Add-on, administrators can configure one high-availability cluster to serve both of these roles simultaneously.

The third tier in the above example does not have to use the Load Balancer Add-on, but failing to use a highly available solution would introduce a critical single point of failure.

4.3. Load Balancer — A Block Diagram

The Load Balancer consists of two main technologies to monitor cluster members and cluster services: Keepalived and HAProxy.

4.3.1. keepalived

The **keepalived** daemon runs on both the active and passive LVS routers. All routers running **keepalived** use the *Virtual Redundancy Routing Protocol* (VRRP). The active router sends VRRP advertisements at periodic intervals; if the backup routers fail to receive these advertisements, a new active router is elected.

On the active router, **keepalived** can also perform load balancing tasks for real servers.

Keepalived is the controlling process related to LVS routers. At boot time, the daemon is started by the **systemctl** command, which reads the configuration file **/etc/keepalived/keepalived.conf**. On the active router, the **keepalived** daemon starts the LVS service and monitors the health of the services based on configured topology. Using VRRP, the active router sends periodic advertisements to the backup routers. On the backup routers, the VRRP instance determines the running status of the active router. If the active router fails advertise after a user-configurable interval, Keepalived initiates failover. During failover, the virtual servers are cleared. The new active router takes control of the VIP, sends out a gratuitous ARP message, sets up IPVS table entries (virtual servers), begins health checks, and starts sending VRRP advertisements.

4.3.2. haproxy

HAProxy offers load balanced services to HTTP and TCP-based services, such as internet-connected services and web-based applications. Depending on the load balancer scheduling algorhithm chosen, **haproxy** is able to process several events on thousands of connections across a pool of multiple real servers acting as one virtual server. The scheduler determines the volume of connections and either assigns them equally in non-weighted schedules or given higher connection volume to servers that can handle higher capacity in weighted algorhithms.

HAProxy allows users to define several proxy services, and performs load balancing services of the traffic for the proxies. Proxies are made up of a frontend and one or more backends. The frontend defines IP address (the VIP) and port the on which the proxy listens, as well as defines the backends to use for a particular proxy.

The backend is a pool of real servers, and defines the load balancing algorithm.

4.4. Load Balancer Scheduling Overview

One of the advantages of using Load Balancer is its ability to perform flexible layer 4 load balancing on the real server pool using Keepalived; or, it can be configured at the application layer (HTTP, HTTPS) using HAProxy. This flexibility is due to the variety of scheduling algorithms an administrator can choose from when configuring Load Balancer Add-On. Load balancing is superior to less flexible methods, such as *Round-Robin DNS* where the hierarchical nature of DNS and the caching by client machines can lead to load imbalances. Additionally, the low-level filtering employed by the LVS router has advantages over application-level request forwarding because balancing loads at the network packet level causes minimal computational overhead and allows for greater scalability.

Using scheduling, the active router can take into account the real servers' activity and, optionally, an administrator-assigned *weight* factor when routing service requests. Using assigned weights gives arbitrary priorities to individual machines. Using this form of scheduling, it is possible to create a group of real servers using a variety of hardware and software combinations and the active router can evenly load each real server.

The scheduling mechanism for Load Balancer is provided by a collection of kernel patches called *IP Virtual Server* or *IPVS* modules. These modules enable *layer 4* (*L4*) transport layer switching, which is designed to work well with multiple servers on a single IP address.

To track and route packets to the real servers efficiently, IPVS builds an *IPVS table* in the kernel. This table is used by the active LVS router to redirect requests from a virtual server address to and returning from real servers in the pool.

4.4.1. Scheduling Algorithms

The structure that the IPVS table takes depends on the scheduling algorithm that the administrator chooses for any given virtual server. To allow for maximum flexibility in the types of services you can cluster and how these services are scheduled, Red Hat Enterprise Linux supports several scheduling algorithms for administrators to choose the right deployment scenario for their services. These algorhithms allow administrators several options for load-balanced deployments as they provide granularity down to specified systems based on their application or computing power, prioritizing of connections to certain servers based on application, and routing based on the individual node's ability to handle client connection volume.

4.4.2. Server Weight and Scheduling

The administrator of Load Balancer can assign a *weight* to each node in the real server pool. This weight is an integer value which is factored into any *weight-aware* scheduling algorithms (such as weighted least-connections) and helps the LVS router more evenly load hardware with different capabilities.

Weights work as a ratio relative to one another. For instance, if one real server has a weight of 1 and the other server has a weight of 5, then the server with a weight of 5 gets 5 connections for every 1 connection the other server gets. The default value for a real server weight is 1.

Although adding weight to varying hardware configurations in a real server pool can help load-balance the cluster more efficiently, it can cause temporary imbalances when a real server is introduced to the real server pool and the virtual server is scheduled using weighted least-connections. For example, suppose there are three servers in the real server pool. Servers A and B are weighted at 1 and the third, server C, is weighted at 2. If server C goes down for any reason, servers A and B evenly distributes the abandoned load. However, once server C comes back online, the LVS router sees it has zero connections and floods the server with all incoming requests until it is on par with servers A and B.

4.5. Routing Methods

Red Hat Enterprise Linux uses *Network Address Translation* (NAT) Routing as well as *Direct Routing* for Load Balancer Add-On, which allows the administrator flexibility when utilizing available hardware and integrating the Load Balancer into an existing network.

4.5.1. NAT Routing

Using NAT routing, the LVS router's public floating IP address and private NAT floating IP address are aliased to two physical network interface controllers (NICs). While it is possible to associate each floating IP address to its own physical device on the LVS router nodes, having more than two NICs is not a requirement.

Using this topology, the active LVS router receives the request and routes it to the appropriate server. The real server then processes the request and returns the packets to the LVS router which uses network address translation to replace the address of the real server in the packets with the LVS router's public VIP address. This process is called *IP masquerading* because the actual IP addresses of the real servers is hidden from the requesting clients.

Using this NAT routing, the real servers may be any kind of machine running various operating systems. The main disadvantage is that the LVS router may become a bottleneck in large cluster deployments because it must process outgoing as well as incoming requests.

4.5.2. Direct Routing

Building a Load Balancer setup that uses direct routing provides increased performance benefits compared to other Load Balancer networking topologies. Direct routing allows the real servers to process and route packets directly to a requesting user rather than passing all outgoing packets through the LVS router. Direct routing reduces the possibility of network performance issues by relegating the job of the LVS router to processing incoming packets only.

In the typical direct routing Load Balancer setup, the LVS router receives incoming server requests through the virtual IP (VIP) and uses a scheduling algorithm to route the request to the real servers. The real server processes the request and sends the response directly to the client, bypassing the LVS router. This method of routing allows for scalability in that real servers can be added without the added burden on the LVS router to route outgoing packets from the real server to the client, which can become a bottleneck under heavy network load.

Upgrading from Red Hat Enterprise Linux High Availability Add-On 6

This appendix provides an overview of upgrading Red Hat Enterprise Linux High Availability Add-On from release 6 to release 7.

A.1. Overview of Differences Between Releases

Red Hat Enterprise Linux 7 High Availability Add-On introduces a new suite of technologies that underlying high-availability technology based on Pacemaker and Corosync that completely replaces the CMAN and RGManager technologies from previous releases of High Availability Add-On. Below are some of the differences between the two releases. For a more comprehensive look at the differences between releases, refer to the appendix titled "Cluster Creation with rgmanager and with Pacemaker" from the *Red Hat Enterprise Linux High Availability Add-On Reference*.

- Configuration Files — Previously, cluster configuration was found in the **`/etc/cluster/cluster.conf`** file, while cluster configuration in release 7 is in **`/etc/corosync/corosync.conf`** for membership and quorum configuration and **`/var/lib/heartbeat/crm/cib.xml`** for cluster node and resource configuration.

- Executable Files — Previously, cluster commands were in **`ccs`** via command-line, **`luci`** for graphical configuration. In Red Hat Enterprise Linux 7 High Availability Add-On, configuration is done via **`pcs`** at the command-line and **`pcsd`** for graphical configuration at the desktop.

- Starting the Service — Previously, all services including those in High Availability Add-On were performed using the **`service`** command to start services and the **`chkconfig`** command to configure services to start upon system boot. This had to be configured separately for all cluster services (**`rgmanager`**, **`cman`**, and **`ricci`**. For example:

  ```
  service rgmanager start
  chkconfig rgmanager on
  ```

  For Red Hat Enterprise Linux 7 High Availability Add-On, the **`systemctl`** controls both manual startup and automated boot-time startup, and all cluster services are grouped in the **`pcsd.service`**. For example:

  ```
  systemctl pcsd.service start
  systemctl pcsd.service on
  pcs cluster start -all
  ```

- User Access — Previously, the root user or a user with proper permissions can access the **`luci`** configuration interface. All access requires the **`ricci`** password for the node.

  In Red Hat Enterprise Linux 7 High Availability Add-On, the **`pcsd`** graphical interface requires that you authenticate as user **`hacluster`**, which is the common system user. The **`root`** user can set the password for **`hacluster`**.

- Creating Clusters, Nodes and Resources — Previously, creation of nodes were performed with the **`ccs`** via command-line or with **`luci`** graphical interface. Creation of a cluster and adding nodes is a separate process. For example, to create a cluster and add a node via command-line, perform the following:

```
ccs -h node1.example.com --createcluster examplecluster
ccs -h node1.example.com --addnode node2.example.com
```

In Red Hat Enterprise Linux 7 High Availability Add-On, adding of clusters, nodes, and resources are done via **pcs** at the command-line, or **pcsd** for graphical configuration. For example, to create a cluster via command-line, perform the following:

```
pcs cluster setup examplecluster node1 node2 ...
```

- Cluster removal — Previously, administrators removed a cluster by deleting nodes manually from the **luci** interface or deleting the **cluster.conf** file from each node

  In Red Hat Enterprise Linux 7 High Availability Add-On, administrators can remove a cluster by issuing the **pcs cluster destroy** command.

Revision History

Revision 0.1-12.405	**Thu Jul 7 2014**	**Rüdiger Landmann**
Add html-single and epub formats		
Revision 0.1-12	**Mon Jun 23 2014**	**John Ha**
Version for 7.0 GA Release		
Revision 0.1-10	**Tue Jun 03 2014**	**John Ha**
Version for 7.0 GA Release		
Revision 0.1-9	**Tue May 13 2014**	**John Ha**
Build for updated version		
Revision 0.1-6	**Wed Mar 26 2014**	**John Ha**
Build for newest draft		
Revision 0.1-4	**Wed Nov 27 2013**	**John Ha**
Build for Beta of Red Hat Enterprise Linux 7		
Revision 0.1-2	**Thu Jun 13 2013**	**John Ha**
First version for Red Hat Enterprise Linux 7		
Revision 0.1-1	**Wed Jan 16 2013**	**Steven Levine**
First version for Red Hat Enterprise Linux 7		

Index