Red Hat Enterprise Linux 7

로드 밸런서 관리

Keepalived 및 HAProxy 구성

Last Updated: 2023-11-18

Red Hat Enterprise Linux 7 로드 밸런서 관리

Keepalived 및 HAProxy 구성

Steven Levine
Red Hat Customer Content Services
slevine@redhat.com

Stephen Wadeley
Red Hat Customer Content Services
swadeley@redhat.com

법적 공지

초록

로드 밸런서 시스템을 구축하면 Keepalived 및 HAProxy를 통해 구성된 라우팅 및 로드 밸런싱 기술을 위해 특수 Linux Virtual Servers(LVS)를 사용하는 프로덕션 서비스에 고가용성 및 확장 가능한 솔루션을 제공합니다. 이 문서에서는 Red Hat Enterprise Linux 7의 Load Balancer 기술을 사용하여 고성능 시스템 및 서비스 구성에 대해 설명합니다.

차례

1장. 로드 밸런서 개요

로드 밸런서는 실제 서버 집합에서 IP 트래픽의 균형을 조정하는 통합 소프트웨어 구성 요소입니다. 클러스터 멤버 및 클러스터 서비스를 모니터링하는 두 가지 주요 기술인 Keepalived 및 HAProxy로 구성됩니다. keepalived는 *Linux 가상 서버* (LVS)를 사용하여 활성 및 수동 라우터에서 부하 분산 및 페일오버 작업을 수행하는 반면 HAProxy는 TCP 및 HTTP 애플리케이션에 대한 부하 분산 및 고가용성 서비스를 수행합니다.

1.1. KEEPALIVED

keepalived 데몬은 활성 및 수동 LVS 라우터 모두에서 실행됩니다. **keepalived** 를 실행하는 모든 라우터는 VRRP(*Virtual Redundancy Routing Protocol*)를 사용합니다. 활성 라우터는 주기적으로 VRRP 알림을 보냅니다. 백업 라우터가 이러한 알림을 수신하지 못하면 새로운 활성 라우터가 선택됩니다.

활성 라우터에서 **keepalived** 는 실제 서버에 대한 부하 분산 작업도 수행할 수 있습니다.

keepalived는 LVS 라우터와 관련된 제어 프로세스입니다. 부팅 시 데몬은 구성 파일 **/etc/keepalived/keepalived.conf** 를 읽는 **systemctl** 명령으로 시작됩니다. 활성 라우터에서 **keepalived** 데몬은 LVS 서비스를 시작하고 구성된 토폴로지를 기반으로 서비스 상태를 모니터링합니다. VRRP를 사용하여 활성 라우터는 주기적인 알림을 백업 라우터에 보냅니다. 백업 라우터에서 VRRP 인스턴스는 활성 라우터의 실행 상태를 결정합니다. 사용자 구성 가능 간격 후 활성 라우터가 알리지 않으면 Keepalived에서 페일오버를 시작합니다. 장애 조치 중에 가상 서버가 지워집니다. 새로운 활성 라우터는 *가상 IP 주소* (VIP)를 제어하고, ARP 메시지를 보내고, IPVS 테이블 항목(가상 서버)을 설정하고, 상태 점검을 시작하고, VRRP 알림 전송을 시작합니다.

Keepalived는 TCP가 연결 기반 데이터 전송을 수행하는 계층 4 또는 전송 계층에서 페일오버를 수행합니다. 실제 서버가 간단한 시간 초과 TCP 연결에 응답하지 못하면 **keepalived** 는 서버가 실패했음을 감지하고 서버 풀에서 제거합니다.

1.2. HAPROXY

HAProxy는 인터넷 연결 서비스 및 웹 기반 애플리케이션과 같은 HTTP 및 TCP 기반 서비스에 부하 분산 서비스를 제공합니다. 선택한 로드 밸런서 스케줄링 알고리즘에 따라 **haproxy** 는 하나의 가상 서버 역할을 하는 여러 실제 서버 풀의 수천 개의 연결에서 여러 이벤트를 처리할 수 있습니다. 스케줄러는 연결 볼륨을 결정하고 가중치가 없는 스케줄에서 동일하게 할당하거나 가중치가 많은 알고리즘에서 더 많은 용량을 처리할 수 있는 서버에 더 높은 연결 볼륨을 할당합니다.

HAProxy를 사용하면 여러 프록시 서비스를 정의하고 프록시에 대한 트래픽의 로드 밸런싱 서비스를 수행할 수 있습니다. 프록시는 프런트 엔드 시스템과 하나 이상의 백엔드 시스템으로 구성됩니다. 프런트 엔드 시스템은 프록시가 수신 대기하는 IP 주소(VIP) 및 포트와 특정 프록시에 사용할 백엔드 시스템을 정의합니다.

백엔드 시스템은 실제 서버 풀이며 부하 분산 알고리즘을 정의합니다.

HAProxy는 계층 7 또는 애플리케이션 계층에서 부하 분산 관리를 수행합니다. 대부분의 경우 관리자는 프로덕션 웹 애플리케이션과 같은 HTTP 기반 로드 밸런싱을 위해 HAProxy를 배포합니다. 이 경우 고가용성 인프라가 비즈니스 연속성에 필요한 부분입니다.

1.3. KEEPALIVED 및 HAPROXY

관리자는 Keepalived 및 HAProxy를 함께 사용하여 보다 강력하고 확장 가능한 고가용성 환경을 제공할 수 있습니다. HAProxy의 속도와 확장성을 사용하여 Keepalived 페일오버 서비스와 함께 HTTP 및 기타 TCP 기반 서비스에 대한 로드 밸런싱을 수행하고, 관리자는 실제 서버에 부하를 분산하고 백업 라우터에 대한 장애 조치를 수행하여 라우터를 사용할 수 없는 경우 연속성을 보장할 수 있습니다.

2장. KEEPALIVED 개요

keepalived는 *활성 LVS 라우터* 와 하나 이상의 선택적 *백업 LVS 라우터에서* 실행됩니다. 활성 LVS 라우터는 다음 두 가지 역할을 합니다.

- 실제 서버에서 부하를 분산합니다.
- 각 실제 서버에서 서비스의 무결성을 확인하려면 다음을 수행하십시오.

활성(마스터) 라우터는 VRRP(*Virtual Router Redundancy Protocol*)를 사용하여 백업 라우터에 활성 상태를 알립니다. 이 경우 마스터 라우터가 정기적으로 알림을 보내야 합니다. 활성 라우터가 알림 전송을 중지하면 새 마스터가 선택됩니다.

참고

Red Hat은 구성이 사용할 VRRP 버전을 변경하는 keepalived의 롤링 업데이트를 지원하지 않습니다. 모든 라우터는 keepalived 로드 밸런서 구성에서 동일한 버전의 VRRP를 실행해야 합니다. VRRP 버전이 일치하지 않으면 다음과 같은 메시지가 표시됩니다.

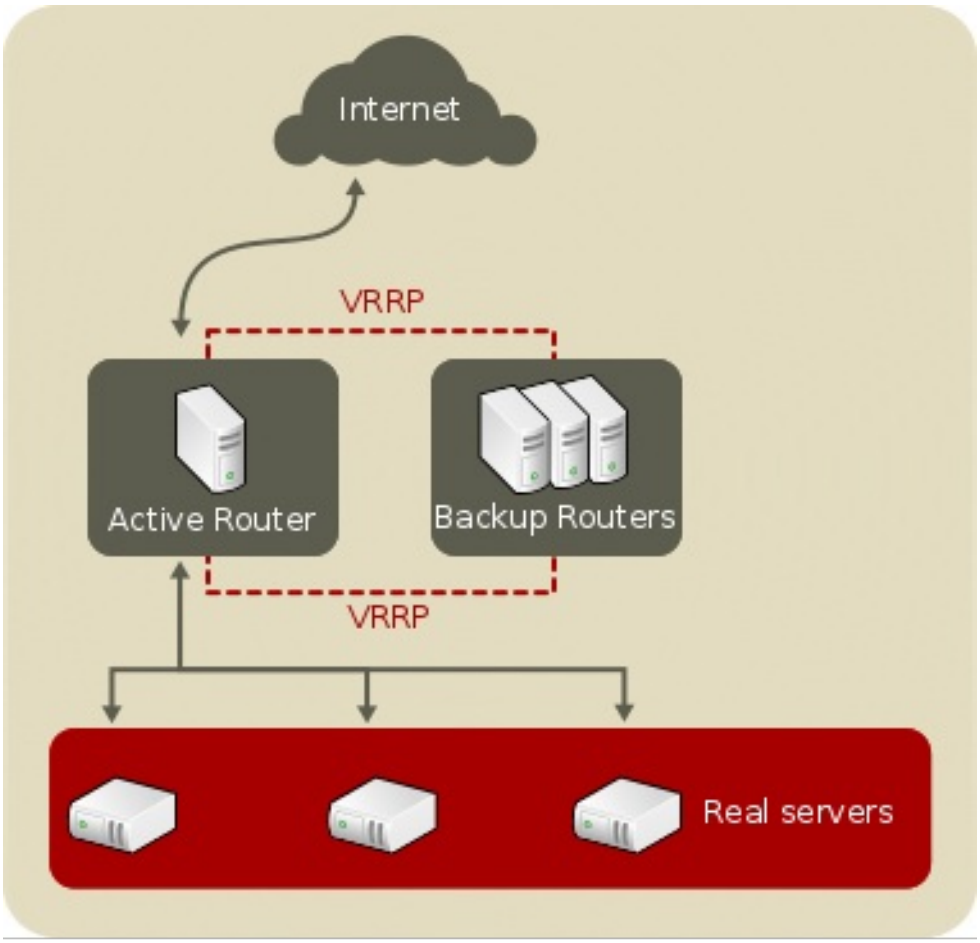
```
Aug  3 17:07:19 hostname Keepalived_vrrp[123]: receive an invalid ip number count
associated with VRID!
Aug  3 17:07:19 hostname Keepalived_vrrp[123]: bogus VRRP packet received on
em2 !!!
Aug  3 17:07:19 hostname Keepalived_vrrp[123]: VRRP_Instance(vrrp_ipv6) ignoring
received advertisment...
```

Red Hat은 모든 시스템에서 동일한 keepalived 버전을 실행해야 하며 호환성 문제를 방지하려면 가능한 경우 keepalived 설정이 동일해야 합니다.

2.1. 기본 KEEPALIVED LOAD BALANCER 구성

그림 2.1. "기본 로드 밸런서 구성" 두 계층으로 구성된 간단한 Keepalived Load Balancer 구성이 표시됩니다. 첫 번째 계층에서는 활성 상태이며 여러 백업 LVS 라우터가 있습니다. 각 LVS 라우터에는 인터넷상의 하나의 인터페이스와 개인 네트워크에 있는 하나의 인터페이스가 있으므로 두 개의 네트워크 간 트래픽을 규제할 수 있습니다. 이 예에서 활성 라우터는 *네트워크 주소 변환* 또는 *NAT* 를 사용하여 인터넷에서 트래픽을 두 번째 계층의 다양한 실제 서버로 보내어 필요한 서비스를 제공합니다. 따라서 이 예제의 실제 서버는 전용 사설 네트워크 세그먼트에 연결되어 모든 공용 트래픽을 활성 LVS 라우터를 통해 전달합니다. 외부 세계에는 서버가 하나의 엔티티로 표시됩니다.

그림 2.1. 기본 로드 밸런서 구성



[D]

LVS 라우터에 도달하는 서비스 요청은 *가상 IP* 주소 또는 *VIP* 로 주소가 지정됩니다. 이는 www.example.com와 같이 정규화된 도메인 이름과 연결하는 사이트 관리자가 공개적으로 라우팅 가능한 주소이며 하나 이상의 *가상 서버에* 할당됩니다. 가상 서버는 특정 가상 IP에서 수신 대기하도록 구성된 서비스입니다. VIP 주소는 장애 조치 중에 하나의 LVS 라우터에서 다른 LVS 라우터로 마이그레이션하므로 해당 IP 주소에 존재함을 유지합니다. VIP는 *유동 IP 주소라고도 합니다*.

VIP 주소는 LVS 라우터를 인터넷에 연결하는 동일한 장치에 할당할 수 있습니다. 예를 들어 **eth0** 이 인터넷에 연결된 경우 여러 가상 서버를 **eth0** 에 할당할 수 있습니다. 또는 각 가상 서버를 서비스당 별도의 장치와 연결할 수 있습니다. 예를 들어, HTTP 트래픽은 192.168.1.111의 **eth0** 에서 처리할 수 있지만 FTP 트래픽을 192.168.1.222의 **eth0** 에서 처리할 수 있습니다.

하나의 활성 라우터와 하나의 수동 라우터가 포함된 배포 시나리오에서는 활성 라우터의 역할은 가상 IP 주소에서 실제 서버로 서비스 요청을 리디렉션하는 것입니다. 리디렉션은 2.3절. “keepalived 스케줄링 개요” 에서 추가로 설명된 8개의 로드 밸런싱 알고리즘 중 하나를 기반으로 합니다.

또한 활성 라우터는 간단한 TCP 연결, HTTP 및 HTTPS의 세 가지 기본 제공 상태 점검을 통해 실제 서버에서 특정 서비스의 전반적인 상태를 동적으로 모니터링합니다. TCP 연결의 경우 활성 라우터는 주기적으로 특정 포트의 실제 서버에 연결할 수 있는지 확인합니다. HTTP 및 HTTPS의 경우 활성 라우터는 주기적으로 실제 서버에서 URL을 가져오고 해당 콘텐츠를 확인합니다.

백업 라우터는 대기 시스템의 역할을 수행합니다. 라우터 페일오버는 VRRP에 의해 처리됩니다. 시작 시 모든 라우터가 멀티 캐스트 그룹에 참여합니다. 이 멀티 캐스트 그룹은 VRRP 알림을 보내고 수신하는 데 사용됩니다. VRRP는 우선 순위 기반 프로토콜이므로 우선 순위가 가장 높은 라우터가 마스터로 선택됩니다. 라우터가 마스터 선택되면 주기적으로 멀티 캐스트 그룹에 VRRP 알림을 보낼 책임이 있습니다.

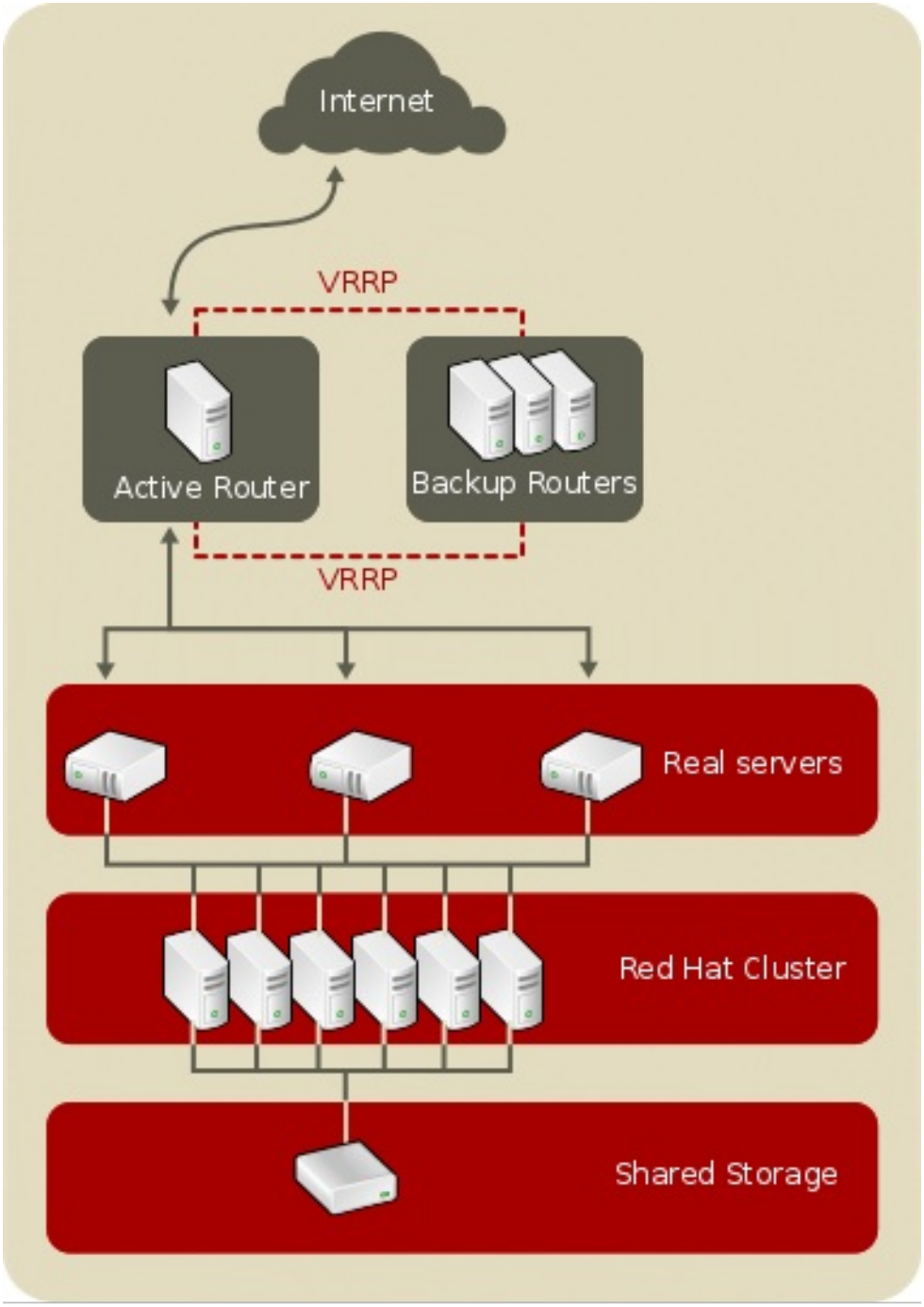
백업 라우터가 특정 기간 내에 알림을 수신하지 못하면 광고 간격에 따라 새 마스터가 선택됩니다. 새 마스터는 VIP를 인수하고 ARP(*Address Resolution Protocol*) 메시지를 보냅니다. 라우터가 활성 서비스로 돌아가면 백업 또는 마스터가 될 수 있습니다. 동작은 라우터의 우선 순위에 따라 결정됩니다.

그림 2.1. "기본 로드 밸런서 구성" 에서 사용되는 간단한 2계층 구성은 정적 웹 페이지와 같이 자주 변경되지 않는 데이터를 제공하는 데 가장 적합합니다. 개별 실제 서버는 각 노드 간에 데이터를 자동으로 동기화하지 않기 때문입니다.

2.2. 3계층 KEEPALIVED LOAD BALANCER 구성

그림 2.2. "3계층 로드 밸런서 구성" 는 일반적인 3계층 Keepalived Load Balancer 토폴로지를 보여줍니다. 이 예에서 활성 LVS 라우터는 인터넷에서 실제 서버 풀로 요청을 라우팅합니다. 그런 다음 각 실제 서버는 네트워크를 통해 공유 데이터 소스에 액세스합니다.

그림 2.2. 3계층 로드 밸런서 구성



[D]

이 구성은 액세스 가능한 데이터가 중앙의 고가용성 서버에 저장되고 내보낸 NFS 디렉토리 또는 Samba 공유를 통해 각 실제 서버에서 액세스하는 사용 중인 FTP 서버에 적합합니다. 이 토폴로지는 트랜잭션을 위해 중앙 고가용성 데이터베이스에 액세스하는 웹 사이트에도 권장됩니다. 또한 관리자는 로드 밸런서와 함께 활성 활성 구성을 사용하여 이러한 두 역할을 동시에 제공하도록 하나의 고가용성 클러스터를 구성할 수 있습니다.

위 예제의 세 번째 계층은 로드 밸런서를 사용할 필요는 없지만 고가용성 솔루션을 사용하지 않으면 중요한 단일 장애 지점이 발생합니다.

2.3. KEEPALIVED 스케줄링 개요

Keepalived를 사용하면 지원되는 다양한 스케줄링 알고리즘으로 인해 실제 서버에서 트래픽을 분산할 때 상당한 유연성을 제공합니다. 로드 밸런싱은 DNS의 계층적 특성과 클라이언트 머신의 캐싱으로 인해 부하 분산이 발생할 수 있는 *Round-Robin DNS* 와 같이 유연성이 떨어집니다. 또한, LVS 라우터에서 사용하는 하위 수준 필터링은 네트워크 패킷 수준에서의 분산 로드로 인해 컴퓨팅 오버헤드가 최소화되고 확장성을 높이기 때문에 애플리케이션 수준 요청 전달보다 이점이 있습니다.

할당된 가중치를 사용하면 개별 시스템에 임의의 우선 순위를 지정할 수 있습니다. 이러한 형태의 스케줄링을 사용하면 다양한 하드웨어 및 소프트웨어 조합을 사용하여 실제 서버 그룹을 생성할 수 있으며 활성 라우터는 각 실제 서버를 균등하게 로드할 수 있습니다.

Keepalived의 스케줄링 메커니즘은 *IP 가상 서버 또는 IP VS* 모듈이라는 커널 패치 컬렉션에서 제공합니다. 이러한 모듈을 사용하면 단일 IP 주소에서 여러 서버에서 원활하게 작동하도록 설계된 *계층4(L* 4) 전송 계층 전환이 가능합니다.

패킷을 실제 서버로 효율적으로 추적하고 라우팅하기 위해 IPVS는 커널에서 *IPVS 테이블을* 빌드합니다. 이 테이블은 활성 LVS 라우터에서 가상 서버 주소로 요청을 리디렉션하고 풀의 실제 서버에서 반환하는 데 사용됩니다.

2.3.1. keepalived Scheduling 알고리즘

IPVS 테이블이 사용하는 구조는 관리자가 지정된 가상 서버에 대해 선택하는 스케줄링 알고리즘에 따라 다릅니다. 클러스터 가능한 서비스 유형 및 이러한 서비스 예약 방법을 최대한 유연하게 지원하기 위해 Keepalived는 아래에 나열된 다음 스케줄링 알고리즘을 지원합니다.

round-Robin Scheduling

실제 서버 풀 주위에 각 요청을 순차적으로 배포합니다. 이 알고리즘을 사용하면 모든 실제 서버가 용량 또는 로드와 관계없이 동일하게 처리됩니다. 이 스케줄링 모델은 라운드 로빈 DNS와 유사하지만 호스트 기반이 아닌 네트워크 연결 기반이므로 더 세분화됩니다. 로드 밸런서 라운드 로빈 스케줄링은 캐시된 DNS 쿼리로 인한 불균형이 발생하지 않습니다.

가중치가 Round-Robin Scheduling

각 요청을 실제 서버 풀에 순차적으로 배포하지만 용량이 큰 서버에 더 많은 작업을 제공합니다. 용량은 사용자가 할당한 가중치 요인으로 표시되므로 동적 로드 정보에 의해 상향 또는 아래로 조정됩니다.

풀의 실제 서버 용량에 상당한 차이가 있는 경우 가중치가 지정된 라운드 로빈 스케줄링을 선택하는 것이 좋습니다. 그러나 요청 부하가 크게 달라지면 가중치가 많은 서버가 요청 공유보다 더 많이 응답할 수 있습니다.

least-Connection

활성 연결이 더 적은 실제 서버에 더 많은 요청을 배포합니다. IPVS 테이블을 통해 실제 서버에 대한 실시간 연결을 추적하므로 최소 연결은 동적 스케줄링 알고리즘의 유형이므로 요청 로드에 높은 수준의 변형이 있는 경우 더 나은 선택이 가능합니다. 각 멤버 노드에 거의 동일한 용량이 있는 실제 서버 풀에

가장 적합합니다. 서버 그룹에 다른 기능이 있는 경우 weighted least-connection 스케줄링을 더 잘 선택할 수 있습니다.

가중치 Least-Connections

용량을 기준으로 활성 연결이 적은 서버에 더 많은 요청을 배포합니다. 용량은 사용자가 할당한 가중치로 표시되므로 동적 로드 정보에 의해 상향 또는 아래로 조정됩니다. 가중치를 추가하면 실제 서버 풀에 다양한 용량의 하드웨어가 포함된 경우 이 알고리즘이 이상적입니다.

locality-Based Least-Connection Scheduling

대상 IP에 비해 활성 연결이 적은 서버에 더 많은 요청을 배포합니다. 이 알고리즘은 프록시 캐시 서버 클러스터에서 사용하도록 설계되었습니다. 서버가 용량을 초과하고 부하에 서버가 있는 경우가 아니면 IP 주소의 IP 주소를 해당 주소의 서버로 라우팅합니다. 이 경우 로드가 가장 적은 실제 서버에 IP 주소를 할당합니다.

복제 일정을 사용한 지역 기반 Least-Connection 스케줄링

대상 IP에 비해 활성 연결이 적은 서버에 더 많은 요청을 배포합니다. 이 알고리즘은 프록시 캐시 서버 클러스터에서 사용하도록 설계되었습니다. 대상 IP 주소를 실제 서버 노드의 하위 집합에 매핑하여 로컬 기반 Least-Connection 스케줄링과 다릅니다. 그런 다음 요청은 이 하위 집합에서 연결 수가 가장 낮은 서버로 라우팅됩니다. 대상 IP의 모든 노드가 용량을 초과하는 경우 실제 서버의 전체 풀에서 해당 대상 IP의 실제 서버 하위 집합에 가장 적은 실제 서버를 추가하여 해당 대상 IP 주소에 대한 새 서버를 복제합니다. 그런 다음 가장 로드된 노드가 실제 서버 하위 집합에서 삭제되어 초과 복제를 방지합니다.

대상 해시 스케줄링

정적 해시 테이블에서 대상 IP를 확인하여 실제 서버 풀에 요청을 배포합니다. 이 알고리즘은 프록시 캐시 서버 클러스터에서 사용하도록 설계되었습니다.

소스 해시 스케줄링

정적 해시 테이블에서 소스 IP를 조회하여 실제 서버 풀에 요청을 배포합니다. 이 알고리즘은 여러 방화벽이 있는 LVS 라우터를 위해 설계되었습니다.

Expected Delay

지정된 서버의 연결 수를 할당된 가중치로 나눈 연결 수에 따라 지연이 예상되는 서버에 연결 요청을 배포합니다.

never Queue

처음 찾고 연결하는 2가지 스케줄러로, 유휴 상태이거나 연결이 없는 서버에 연결 요청을 보냅니다. 유휴 서버가 없는 경우 스케줄러는 *Shortest Expected Delay* 와 동일한 방식으로 지연이 가장 적은 서버로 기본 설정됩니다.

2.3.2. 서버 Weight 및 스케줄링

로드 밸런서 관리자는 실제 서버 풀의 각 노드에 *가중치* 를 할당할 수 있습니다. 이 가중치는 임의의 가중치 *인식 스케줄링 알고리즘(예: 가중치* 최소 연결)에 영향을 미치는 정수 값이며, LVS 라우터가 다른 기능을 사용하여 하드웨어를 균등하게 로드하는 데 도움이 됩니다.

가중치는 서로 상대적인 비율로 작동합니다. 예를 들어 하나의 실제 서버에 가중치가 1이고 다른 서버의 가중치가 5인 경우 가중치가 5인 서버는 다른 서버가 얻는 1개의 연결에 대해 5개의 연결을 가져옵니다. 실제 서버 가중치의 기본값은 1입니다.

실제 서버 풀의 다양한 하드웨어 구성에 가중치를 추가하면 클러스터를 보다 효율적으로 분산하는 데 도움이 될 수 있지만 실제 서버가 실제 서버 풀에 도입되고 가상 서버가 가중치 최소 연결을 사용하여 예약될 때 일시적인 불균형이 발생할 수 있습니다. 예를 들어 실제 서버 풀에 세 개의 서버가 있다고 가정합니다. 서버 A 및 B의 가중치는 1이고 세 번째 서버 C의 가중치는 2입니다. 어떤 이유로든 서버 C가 다운되면 서버 A 및 B는 중단된 로드를 균등하게 분산합니다. 그러나 서버 C가 다시 온라인 상태가 되면 LVS 라우터에는 0개의 연결이 있고 서버 A 및 B와 대조될 때까지 들어오는 모든 요청이 있는 서버를 플러딩합니다.

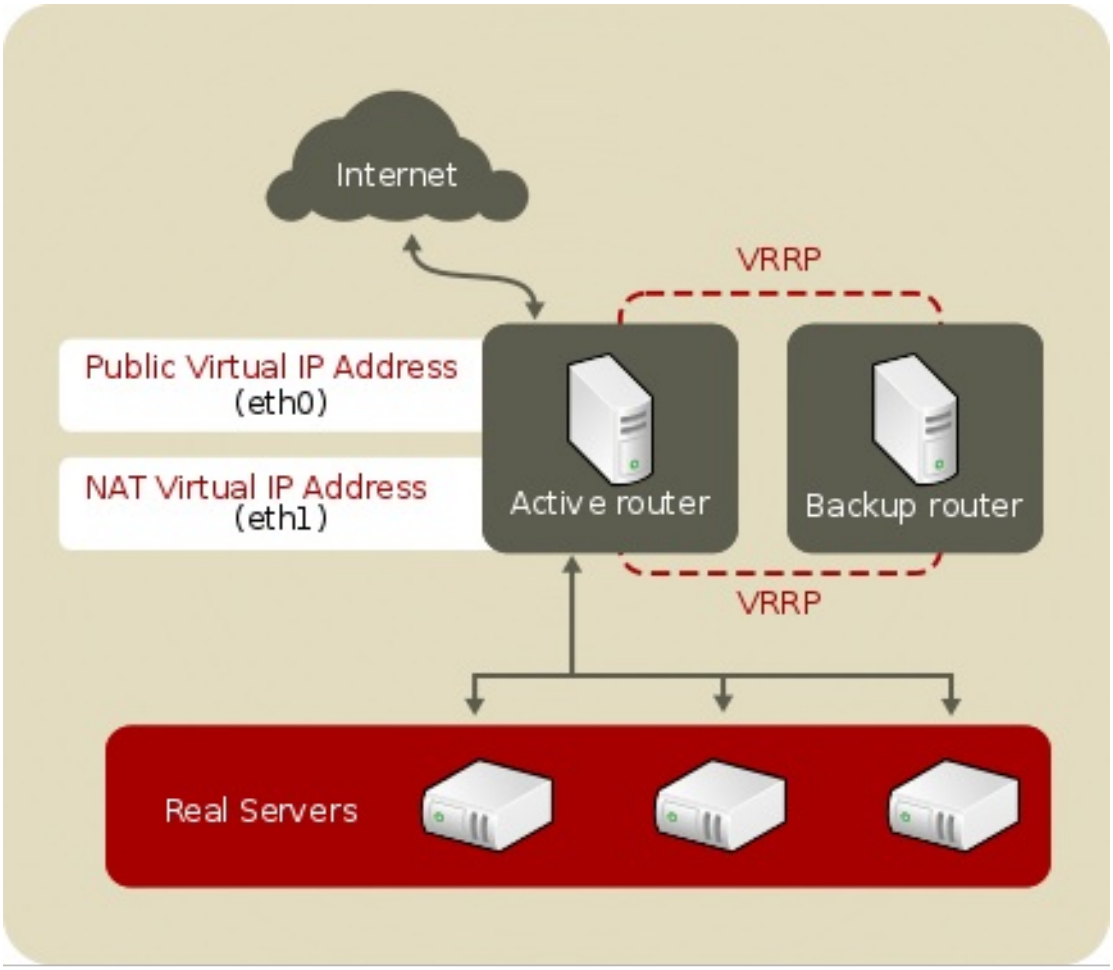
2.4. 라우팅 방법

Red Hat Enterprise Linux는 Keepalived에*NAT*(*Network Address Translation*) 또는 직접 라우팅을 사용합니다. 따라서 사용 가능한 하드웨어를 활용하고 로드 밸런서를 기존 네트워크에 통합할 때 관리자가 상당한 유연성을 확보할 수 있습니다.

2.4.1. NAT 라우팅

그림 2.3. "NAT 라우팅을 사용하여 구현됨" 에서는 NAT 라우팅을 사용하여 인터넷과 사설 네트워크 간에 요청을 이동하는 로드 밸런서를 보여줍니다.

그림 2.3. NAT 라우팅을 사용하여 구현됨



[D]

이 예제에는 활성 LVS 라우터에는 두 개의 NIC가 있습니다. 인터넷의 NIC에는 eth0에 *실제 IP 주소* 와 유동 IP 주소가 있습니다. 사설 네트워크 인터페이스의 NIC에는 실제 IP 주소와 eth1에 유동 IP 주소가 있습니다. 페일오버가 발생하는 경우 인터넷에 직면하는 가상 인터페이스와 개인 직면의 가상 인터페이스가 백업 LVS 라우터에 의해 동시에 수행됩니다. 사설 네트워크에 있는 모든 실제 서버는 NAT 라우터의 유동 IP를 기본 경로로 사용하여 활성 LVS 라우터와 통신하여 인터넷의 요청에 응답할 수 있는 기능이 손상되지 않습니다.

이 예에서는 LVS 라우터의 공용 유동 IP 주소와 개인 NAT 유동 IP 주소가 물리적 NIC에 할당됩니다. 각 유동 IP 주소를 LVS 라우터 노드의 자체 물리적 장치에 연결할 수 있지만 NIC를 두 개 이상 보유하는 것은 필수 사항은 아닙니다.

이 토폴로지를 사용하면 활성 LVS 라우터에서 요청을 수신하고 적절한 서버로 라우팅합니다. 그런 다음 실제 서버에서 요청을 처리하고 네트워크 주소 변환을 사용하는 LVS 라우터에 패킷을 반환하여 패킷의 실제 서버 주소를 LVS 라우터의 공용 VIP 주소로 바꿉니다. 이 프로세스를 실제 서버의 실제 *IP 주소가 요청 클라이언트로부터 숨겨져 있기 때문에 IP 마스커레이딩* 이라고 합니다.

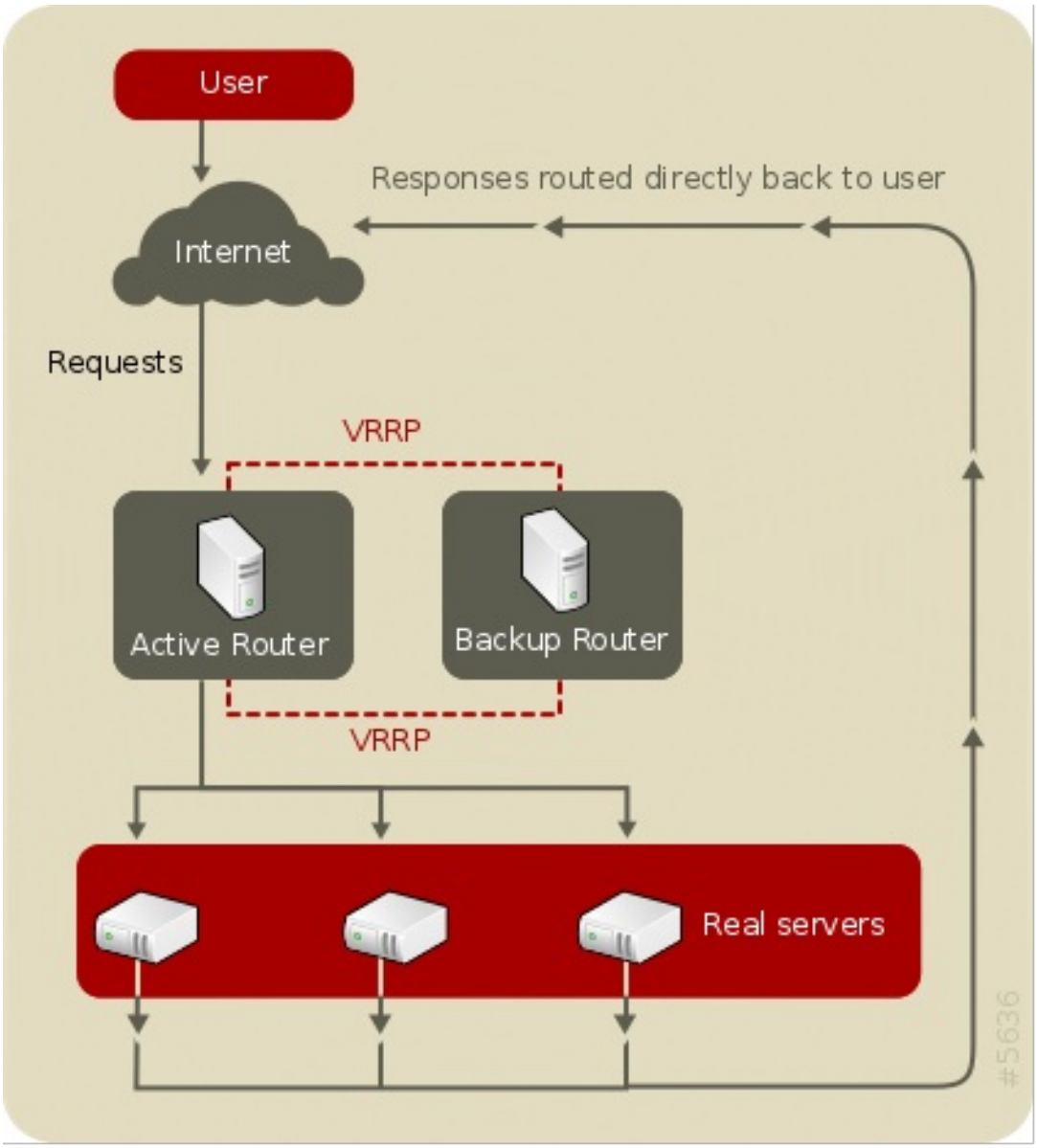
이 NAT 라우팅을 사용하면 실제 서버가 다양한 운영 체제를 실행하는 모든 종류의 시스템일 수 있습니다. 주요 단점은 LVS 라우터가 들어오는 요청과 들어오는 요청을 처리해야 하므로 대규모 클러스터 배포에서 병목 현상이 발생할 수 있다는 것입니다.

ipvs 모듈은 iptables 및 ip6tables NAT와 독립적인 자체 내부 NAT 루틴을 사용합니다. 그러면 **/etc/keepalived/keepalived.conf** 파일의 DR과 달리 실제 서버가 NAT에 대해 구성된 경우 IPv4 및 IPv6 NAT가 원활하게 수행됩니다.

2.4.2. 직접 라우팅

직접 라우팅을 사용하는 로드 밸런서 설정을 구축하면 다른 로드 밸런서 네트워킹 토폴로지에 비해 성능상의 이점이 향상됩니다. 직접 라우팅을 사용하면 실제 서버에서 LVS 라우터를 통해 나가는 모든 패킷을 전달하는 대신 요청하는 사용자에게 직접 패킷을 처리하고 라우팅할 수 있습니다. 직접 라우팅은 들어오는 패킷만 처리하기 위해 LVS 라우터의 작업을 다시 부여하여 네트워크 성능 문제가 발생할 가능성을 줄입니다.

그림 2.4. 직접 라우팅으로 구현되는 로드 밸런서



[D]

일반적인 직접 라우팅 로드 밸런서 설정에서 LVS 라우터는 VIP(가상 IP)를 통해 들어오는 서버 요청을 수신하고 스케줄링 알고리즘을 사용하여 요청을 실제 서버로 라우팅합니다. 실제 서버는 요청을 처리하고 LVS 라우터를 우회하여 응답을 클라이언트로 직접 보냅니다. 이 라우팅 방법을 사용하면 LVS 라우터의 추가 부담 없이 실제 서버의 확장성을 추가하여 실제 서버에서 클라이언트로 나가는 패킷을 라우팅할 수 있으며, 이는 네트워크 로드가 많은 병목 현상이 될 수 있습니다.

2.4.2.1. 직접 라우팅 및 ARP 제한

로드 밸런서에서 직접 라우팅을 사용하는 데는 많은 이점이 있지만 제한 사항도 있습니다. 직접 라우팅을 통한 로드 밸런서의 가장 일반적인 문제는ARP(*Address Resolution Protocol*)입니다.

일반적인 상황에서 인터넷의 클라이언트는 IP 주소로 요청을 보냅니다. 네트워크 라우터는 일반적으로 IP 주소와 관련하여 ARP가 있는 시스템의 MAC 주소로 요청을 대상에 보냅니다. ARP 요청은 네트워크의 모든 연결된 시스템에 브로드캐스트되며 올바른 IP/MAC 주소 조합이 있는 시스템은 패킷을 수신합니다. IP/MAC 연결은 ARP 캐시에 저장되며 주기적으로 지워지고 (일반적으로 15분마다) IP/MAC 연결로 다시 채워집니다.

직접 라우팅 로드 밸런서 설정에서 ARP 요청의 문제는 요청을 처리하기 위해 IP 주소에 대한 클라이언트

요청이 MAC 주소와 연결되어야 하므로 로드 밸런서 시스템의 가상 IP 주소도 MAC에 연결되어야 한다는 것입니다. 그러나 LVS 라우터와 실제 서버 모두 동일한 VIP를 사용하므로 ARP 요청은 VIP와 연결된 모든 머신에 브로드캐스트됩니다. 이로 인해 VIP가 실제 서버 및 처리 요청 중 하나에 직접 연결되어 LVS 라우터를 완전히 우회하고 로드 밸런서 설정의 목적을 손상시키는 것과 같은 여러 문제가 발생할 수 있습니다.

이 문제를 해결하려면 들어오는 요청이 항상 실제 서버 중 하나가 아닌 LVS 라우터로 전송되었는지 확인합니다. 이 작업은 ARP 요청을 필터링하거나 IP 패킷을 필터링하여 수행할 수 있습니다. ARP 필터링은 **arptables** 유틸리티 및 IP 패킷을 **iptables** 또는 **firewalld** 를 사용하여 필터링할 수 있습니다. 두 가지 접근 방식은 다음과 같습니다.

- ARP 필터링 방법은 실제 서버에 도달하는 요청을 차단합니다. 이렇게 하면 ARP가 VIP를 실제 서버와 연결하지 못하도록 활성 가상 서버가 MAC 주소에 응답합니다.
- IP 패킷 필터링 방법을 사용하면 다른 IP 주소가 있는 실제 서버로 패킷을 라우팅할 수 있습니다. 이는 먼저 실제 서버에서 VIP를 구성하지 않고 ARP 문제를 완전히 처리합니다.

2.5. 지속성 및 방화벽 표시 KEEPALIVED로 표시

특정 상황에서는 클라이언트가 최상의 사용 가능한 서버로 해당 요청을 전송하는 대신 동일한 실제 서버에 반복적으로 다시 연결하는 것이 바람직할 수 있습니다. 이러한 상황의 예로는 다중 화면 웹 양식, 쿠키, SSL 및 FTP 연결이 있습니다. 이러한 경우 컨텍스트를 유지하기 위해 동일한 서버에서 트랜잭션을 처리하지 않는 한 클라이언트가 제대로 작동하지 않을 수 있습니다. keepalived는 *지속성* 및 *방화벽 표시* 라는 두 가지 다른 기능을 제공합니다.

2.5.1. 지속성

활성화하면 지속성이 타이머처럼 작동합니다. 클라이언트가 서비스에 연결하면 Load Balancer는 지정된 기간 동안 마지막 연결을 기억합니다. 동일한 클라이언트 IP 주소가 해당 기간 내에 다시 연결하면 이전에 연결된 동일한 서버로 전송되어 부하 분산 메커니즘을 우회합니다. 시간 창 외부에서 연결이 발생하면 스케줄링 규칙에 따라 처리됩니다.

또한 지속성을 통해 관리자는 클라이언트 IP 주소 테스트에 더 높은 수준의 지속성이 있는 주소를 제어하기 위한 툴로 클라이언트 IP 주소 테스트에 적용할 서브넷 마스크를 지정할 수 있으므로 해당 서브넷에 대한 연결을 그룹화할 수 있습니다.

서로 다른 포트로 향하는 연결을 그룹화하는 것은 FTP와 같이 두 개 이상의 포트를 사용하는 프로토콜에 중요할 수 있습니다. 그러나 지속성은 다른 포트를 대상으로 하는 연결을 그룹화하는 문제를 해결하는 가장 효율적인 방법은 아닙니다. 이러한 경우 *방화벽 표시*를 사용하는 것이 가장 좋습니다.

2.5.2. 방화벽 마크

방화벽 마크는 프로토콜 또는 관련 프로토콜 그룹에 사용되는 포트를 그룹화하는 쉽고 효율적인 방법입니다. 예를 들어, 로드 밸런서가 전자 상거래 사이트를 실행하도록 배포되면 방화벽 마크를 사용하여 포트 80에 HTTP 연결을 번들하고 포트 443에 HTTPS 연결을 보호할 수 있습니다. 각 프로토콜에 대해 동일한 방화벽 마크를 가상 서버에 할당하면 LVS 라우터가 연결이 열린 후 모든 요청을 동일한 실제 서버로 전달하므로 트랜잭션에 대한 상태 정보를 유지할 수 있습니다.

효율성과 사용 편의성 때문에 로드 밸런서 관리자는 연결을 그룹화할 수 있을 때마다 지속성 대신 방화벽 표시를 사용해야 합니다. 그러나 관리자는 방화벽 표시와 함께 가상 서버에 지속성을 추가하여 적절한 기간 동안 클라이언트를 동일한 서버에 다시 연결해야 합니다.

3장. KEEPALIVED의 로드 밸런서 사전 요구 사항 설정

keepalived 를 사용하는 로드 밸런서는 LVS 라우터와 실제 서버의 두 가지 기본 그룹으로 구성됩니다. 단일 장애 지점을 방지하려면 각 그룹에 두 개 이상의 멤버가 있어야 합니다.

LVS 라우터 그룹은 Red Hat Enterprise Linux를 실행하는 두 개의 동일하거나 매우 유사한 시스템으로 구성되어야 합니다. 하나는 활성 LVS 라우터 역할을 하는 반면 다른 하나는 핫 대기 모드에 남아 있으므로 가능한 동일한 기능을 가깝게 유지해야 합니다.

실제 서버 그룹에 대한 하드웨어를 선택하고 구성하기 전에 사용할 로드 밸런서 토폴로지를 결정합니다.

3.1. NAT 로드 밸런서 네트워크

NAT 토폴로지를 사용하면 기존 하드웨어를 사용할 수 있지만 모든 패킷이 로드 밸런서 라우터를 통과하고 풀에서 나가기 때문에 대용량 로드를 처리할 수 있는 기능이 제한됩니다.

네트워크 레이아웃

NAT 라우팅을 사용하는 로드 밸런서의 토폴로지는 공용 네트워크에 대한 하나의 액세스 지점만 필요하므로 네트워크 레이아웃 관점에서 가장 쉽게 구성할 수 있습니다. 실제 서버는 사설 네트워크에 있으며 LVS 라우터를 통해 모든 요청에 응답합니다.

하드웨어

NAT 토폴로지에서는 각 실제 서버는 LVS 라우터에만 응답하므로 하나의 NIC만 있으면 됩니다. 반면 LVS 라우터는 두 네트워크 간에 트래픽을 라우팅하려면 각각 두 개의 NIC가 필요합니다. 이 토폴로지는 LVS 라우터에서 네트워크 병목 현상을 생성하므로 각 LVS 라우터에서 Gigabit 이더넷 NIC를 사용하여 LVS 라우터에서 처리할 수 있는 대역폭을 늘릴 수 있습니다. Gigabit 이더넷이 LVS 라우터에서 사용되는 경우 실제 서버를 LVS 라우터에 연결하는 모든 스위치에는 부하를 효율적으로 처리하기 위해 두 개 이상의 Gigabit 이더넷 포트가 있어야 합니다.

소프트웨어

NAT 토폴로지는 일부 구성에 **iptables** 를 사용해야 하므로 Keepalived 외부에 많은 양의 소프트웨어 구성이 있을 수 있습니다. 특히 FTP 서비스와 방화벽 마크를 사용하려면 요청을 올바르게 라우팅하기 위해 LVS 라우터를 수동으로 구성해야 합니다.

3.1.1. NAT를 사용하여 로드 밸런서에 대한 네트워크 인터페이스 구성

NAT를 사용하여 로드 밸런서를 설정하려면 먼저 LVS 라우터에서 공용 네트워크와 사설 네트워크의 네트워크 인터페이스를 구성해야 합니다. 이 예에서 LVS 라우터의 공용 인터페이스(**eth0**)는 203.0.113.0/24 네트워크에 있고 실제 서버(**eth1**)에 연결되는 개인 인터페이스는 10.11.12.0/24 네트워크에 있습니다.

중요

작성 시 **NetworkManager** 서비스가 로드 밸런서와 호환되지 않습니다. 특히 SLAAC에서 IPv6 주소를 할당할 때 IPv6 VIP가 작동하지 않는 것으로 알려져 있습니다. 이러한 이유로 여기에 표시된 예제에서는 구성 파일과 **네트워크** 서비스를 사용합니다.

활성 또는 *기본* LVS 라우터 노드에서 공용 인터페이스의 네트워크 구성 파일인 **/etc/sysconfig/network-scripts/ifcfg-eth0** 은 다음과 같습니다.

```
DEVICE=eth0
BOOTPROTO=static
```

```
ONBOOT=yes
IPADDR=203.0.113.9
NETMASK=255.255.255.0
GATEWAY=203.0.113.254
```

LVS 라우터의 개인 NAT 인터페이스에 대한 구성 파일 **/etc/sysconfig/network-scripts/ifcfg-eth1** 은 다음과 같습니다.

```
DEVICE=eth1
BOOTPROTO=static
ONBOOT=yes
IPADDR=10.11.12.9
NETMASK=255.255.255.0
```

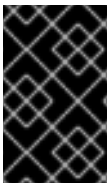
VIP 주소는 정적 주소와 달라야 하지만 범위는 동일합니다. 이 예에서 LVS 라우터의 공용 인터페이스에 대한 VIP를 203.0.113.10으로 구성할 수 있으며 개인 인터페이스의 VIP는 10.11.12.10일 수 있습니다. VIP 주소는 **/etc/keepalived/keepalived.conf** 파일의 **virtual_ipaddress** 옵션에 의해 설정됩니다. 자세한 내용은 4.1절. "기본 Keepalived 구성"의 내용을 참조하십시오.

또한 실제 서버가 NAT 인터페이스의 VIP로 요청을 다시 라우팅하는지 확인합니다.

중요

이 섹션의 샘플 이더넷 인터페이스 구성 설정은 유동 IP 주소가 *아닌* LVS 라우터의 실제 IP 주소를 위한 것입니다.

기본 LVS 라우터 노드의 네트워크 인터페이스를 구성한 후 백업 LVS 라우터의 실제 네트워크 인터페이스를 구성합니다(네트워크의 다른 IP 주소와 충돌하지 않도록 주의하십시오).



중요

백업 노드의 각 인터페이스가 기본 노드의 인터페이스와 동일한 네트워크를 제공하는지 확인합니다. 예를 들면 다음과 같습니다. eth0 기본 노드의 공용 네트워크에 연결하므로 백업 노드의 공용 네트워크에도 연결해야 합니다.

3.1.2. 실제 서버에서 라우팅

NAT 토폴로지에서 실제 서버 네트워크 인터페이스를 구성할 때 가장 중요한 것은 LVS 라우터의 NAT 유동 IP 주소에 대한 게이트웨이를 설정하는 것입니다. 이 예에서 이 주소는 10.11.12.10입니다.

참고

네트워크 인터페이스가 실제 서버에 가동되면 시스템은 공용 네트워크에 ping하거나 연결할 수 없습니다. 이는 정상입니다. 그러나 LVS 라우터의 개인 인터페이스(이 경우 10.11.12.9)의 실제 IP를 ping할 수 있습니다.

실제 서버의 구성 파일 **/etc/sysconfig/network-scripts/ifcfg-eth0** 에서는 다음과 유사합니다.

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
```

```
IPADDR=10.11.12.1
NETMASK=255.255.255.0
GATEWAY=10.11.12.10
```

주의

실제 서버에 **GATEWAY=** 행을 사용하여 두 개 이상의 네트워크 인터페이스가 구성된 경우 첫 번째 서버에서 게이트웨이를 가져옵니다. 따라서 **eth0** 및 **eth1** 이 모두 로드 밸런서에 사용되고 **eth1** 이 로드 밸런서에 사용되는 경우 실제 서버가 요청을 올바르게 라우팅하지 못할 수 있습니다.

/etc/sysconfig/network-scripts/ 디렉터리 내의 네트워크 구성 파일에 **ONBOOT=no** 를 설정하거나 게이트웨이가 먼저 표시되는 인터페이스에 올바르게 설정되었는지 확인하여 불필요한 네트워크 인터페이스를 끄는 것이 가장 좋습니다.

3.1.3. LVS 라우터에서 NAT 라우팅 활성화

각 클러스터형 서비스가 포트 80에서 HTTP와 같은 하나의 포트만 사용하는 간단한 NAT 로드 밸런서 구성에서 관리자는 요청을 외부 환경과 실제 서버 간에 올바르게 라우팅하기 위해 LVS 라우터의 패킷 전달만 활성화해야 합니다. 그러나 클러스터형 서비스에 사용자 세션 중에 동일한 실제 서버로 이동하기 위해 두 개 이상의 포트가 필요한 경우 더 많은 구성이 필요합니다.

LVS 라우터에서 전달이 활성화되고 실제 서버가 설정되고 클러스터형 서비스가 실행되면 **keepalived** 를 사용하여 IP 정보를 구성합니다.

주의

네트워크 구성 파일을 수동으로 편집하거나 네트워크 구성 툴을 사용하여 **eth0** 또는 **eth1** 의 유동 IP를 구성하지 마십시오. 대신 **keepalived.conf** 파일을 사용하여 구성합니다.

완료되면 **keepalived** 서비스를 시작합니다. 활성 LVS 라우터가 가동되어 실행되면 실제 서버 풀에 대한 라우팅 요청을 시작합니다.

3.2. 직접 라우팅을 사용하는 로드 밸런서

직접 라우팅을 사용하면 실제 서버에서 LVS 라우터를 통해 발신 패킷을 전달하는 대신 요청하는 사용자에게 직접 패킷을 처리하고 라우팅할 수 있습니다. 직접 라우팅을 사용하려면 실제 서버가 LVS 라우터를 사용하여 네트워크 세그먼트에 물리적으로 연결되어 나가는 패킷도 처리하고 직접 보낼 수 있어야 합니다.

네트워크 레이아웃

직접 라우팅 로드 밸런서 설정에서 LVS 라우터는 들어오는 요청을 수신하고 처리를 위해 적절한 실제 서버로 라우팅해야 합니다. 그런 다음 실제 서버는 응답을 클라이언트에 *직접* 라우팅해야 합니다. 예를 들어 클라이언트가 인터넷에 있고 LVS 라우터를 통해 패킷을 실제 서버로 보내는 경우 실제 서버는 인

터넷을 통해 클라이언트에 직접 연결할 수 있어야 합니다. 이 작업은 실제 서버가 패킷을 인터넷에 전달할 수 있도록 게이트웨이를 구성하여 수행할 수 있습니다. 서버 풀의 각 실제 서버는 자체 별도의 게이트웨이(및 인터넷에 대한 자체 연결이 있는 각 게이트웨이)를 가질 수 있으므로 처리량과 확장성을 극대화할 수 있습니다. 그러나 일반적인 로드 밸런서 설정의 경우 실제 서버는 하나의 게이트웨이(및 하나의 네트워크 연결)를 통해 통신할 수 있습니다.

하드웨어

직접 라우팅을 사용하는 로드 밸런서 시스템의 하드웨어 요구 사항은 다른 로드 밸런서 토폴로지와 유사합니다. LVS 라우터는 들어오는 요청을 처리하고 실제 서버에 대한 로드 밸런싱을 수행하기 위해 Red Hat Enterprise Linux를 실행해야 하지만 실제 서버가 제대로 작동하려면 Linux 시스템일 필요가 없습니다. LVS 라우터에는 각각 하나 또는 두 개의 NIC가 필요합니다(백업 라우터가 있는 경우 에 따라 다름). 두 개의 NIC를 사용하여 쉽게 구성하고 트래픽을 분리할 수 있습니다. 수신 요청은 NIC에 의해 처리되고 다른 하나는 실제 서버로 라우팅됩니다.

실제 서버는 LVS 라우터를 무시하고 발신 패킷을 클라이언트에 직접 전송하므로 인터넷에 대한 게이트웨이가 필요합니다. 최대 성능과 가용성을 위해 각 실제 서버는 클라이언트가 연결된 네트워크에 대한 자체 전용 연결이 있는 별도의 게이트웨이(예: 인터넷 또는 인트라넷)에 연결할 수 있습니다.

소프트웨어

keepalived 외부 몇 가지 구성이 있으며 특히 직접 라우팅하여 Load Balancer를 사용할 때 ARP 문제에 직면하는 관리자에게 필요합니다. 자세한 내용은 3.2.1절. "arptables를 사용한 직접 라우팅" 또는 3.2.3절. "iptables를 사용한 직접 라우팅" 을 참조하십시오.

3.2.1. arptables를 사용한 직접 라우팅

arptables 를 사용하여 직접 라우팅을 구성하려면 각 서버에 가상 IP 주소가 구성되어 있어야 패킷을 직접 라우팅할 수 있습니다. VIP에 대한 ARP 요청은 실제 서버에서 완전히 무시되며, VIP를 포함하는 전송될 수 있는 ARP 패킷은 VIP 대신 실제 서버의 IP를 포함하도록 조작됩니다.

애플리케이션은 **arptables** 메서드를 사용하여 실제 서버가 서비스하는 각 개별 VIP 또는 포트에 바인딩할 수 있습니다. 예를 들어 **arptables** 메서드를 사용하면 Apache HTTP Server의 여러 인스턴스를 실행하고 시스템의 다른 VIP에 명시적으로 바인딩할 수 있습니다.

그러나 표준 Red Hat Enterprise Linux 시스템 구성 툴을 사용하여 부팅 시 시작되도록 VIP를 구성할 수 없습니다.

각 가상 IP 주소에 대한 ARP 요청을 무시하도록 각 실제 서버를 구성하려면 다음 단계를 수행합니다.

1. **각 실제 서버에서 각 가상 IP 주소에 대한 ARP 테이블 항목을 만듭니다. director가 실제 서버와 통신하는 데 사용하는 IP는 *real_ip* 입니다. 종종 eth0에 바인딩된 IP입니다.**

   ```
   arptables -A IN -d <virtual_ip> -j DROP
   arptables -A OUT -s <virtual_ip> -j mangle --mangle-ip-s <real_ip>
   ```

   이렇게 하면 실제 서버가 가상 IP 주소에 대한 모든 ARP 요청을 무시하고, 대신 서버의 실제 IP를 포함하도록 발신 ARP 응답을 변경합니다. VIP에 대한 ARP 요청에 응답해야 하는 유일한 노드는 현재 활성 LVS 노드입니다.

2. **각 실제 서버에서 이 작업이 완료되면 각 실제 서버에서 다음 명령을 입력하여 ARP 테이블 항목을 저장합니다.**

   arptables-save > /etc/sysconfig/arptables

   systemctl enable arptables.service

systemctl enable 명령을 사용하면 네트워크가 시작되기 전에 시스템이 부팅 시 arptables 구성을 다시 로드합니다.

3. IP 별칭을 만들기 위해 **ip addr** 를 사용하여 모든 실제 서버에서 가상 IP 주소를 구성합니다. 예를 들면 다음과 같습니다.

```
# ip addr add 192.168.76.24 dev eth0
```

4. Direct Routing에 Keepalived를 구성합니다. 이 작업은 **keepalived.conf** 파일에 ***lb_kind DR*** 를 추가하여 수행할 수 있습니다. 자세한 내용은 *4장. Keepalived를 사용한 초기 로드 밸런서 구성*을 참조하십시오.

3.2.2. firewalld를 사용한 직접 라우팅

firewalld 를 사용하여 방화벽 규칙을 생성하여 직접 라우팅 방법을 사용하여 ARP 문제를 해결할 수도 있습니다. **firewalld** 를 사용하여 직접 라우팅을 구성하려면 실제 서버가 시스템에 VIP 주소가 없는 경우에도 실제 서버가 VIP 주소로 전송된 패킷을 서비스하도록 투명한 프록시를 생성하는 규칙을 추가해야 합니다.

firewalld 방법은 **arptables** 메서드보다 쉽게 구성할 수 있습니다. 가상 IP 주소 또는 주소는 활성 LVS director에만 있으므로 이 방법은 LVS ARP 문제를 완전히 우회합니다.

그러나 모든 반환 패킷을 전달하는 데 오버헤드가 있으므로 **firewalld** 방법을 사용하여 **arptables** 를 사용하는 성능 문제가 있습니다.

firewalld 방법을 사용하여 포트를 재사용할 수도 없습니다. 예를 들어 둘 다 가상 IP 주소 대신 ***INADDR_ANY*** 에 바인딩해야 하므로 포트 80에 바인딩된 두 개의 별도의 Apache HTTP Server 서비스를 실행할 수 없습니다.

firewalld 방법을 사용하여 직접 라우팅을 구성하려면 모든 실제 서버에서 다음 단계를 수행합니다.

1. **firewalld** 가 실행 중인지 확인합니다.

```
# systemctl start firewalld
```

시스템을 시작할 때 **firewalld** 가 활성화되어 있는지 확인합니다.

```
# systemctl enable firewalld
```

2. 실제 서버에 서비스를 제공하기 위해 모든 VIP, 포트, 프로토콜(TCP 또는 UDP) 조합에 대해 다음 명령을 입력합니다. 이 명령을 실행하면 실제 서버가 제공된 VIP 및 포트로 향하는 패킷을 처리합니다.

```
# firewall-cmd --permanent --direct --add-rule ipv4 nat PREROUTING 0 -d vip -p tcp|udp -m tcp|udp --dport port -j REDIRECT
```

3. 방화벽 규칙을 다시 로드하고 상태 정보를 유지합니다.

```
# firewall-cmd --reload
```

현재 영구 구성은 새로운 **firewalld** 런타임 구성과 다음 시스템 시작 시 구성이 됩니다.

3.2.3. iptables를 사용한 직접 라우팅

iptables 방화벽 규칙을 생성하여 직접 라우팅 방법을 사용하여 ARP 문제를 해결할 수도 있습니다. **iptables** 를 사용하여 직접 라우팅을 구성하려면 실제 서버가 시스템에 VIP 주소가 없는 경우에도 실제 서버가 VIP 주소로 전송된 패킷을 서비스하도록 투명한 프록시를 생성하는 규칙을 추가해야 합니다.

iptables 방법은 **arptables** 메서드보다 쉽게 구성할 수 있습니다. 또한 가상 IP 주소는 활성 LVS director에만 있으므로 이 방법은 LVS ARP 문제를 완전히 우회합니다.

그러나 모든 패킷을 전달하는 데 오버헤드가 있으므로 **iptables** 방법을 사용하여 **arptables** 를 사용하는 성능 문제가 있습니다.

iptables 방법을 사용하여 포트를 재사용할 수도 없습니다. 예를 들어 둘 다 가상 IP 주소 대신 ***INADDR_ANY*** 에 바인딩해야 하므로 포트 80에 바인딩된 두 개의 별도의 Apache HTTP Server 서비스를 실행할 수 없습니다.

iptables 방법을 사용하여 직접 라우팅을 구성하려면 다음 단계를 수행합니다.

1. 각 실제 서버에서 실제 서버에 대해 서비스를 제공하기 위해 모든 VIP, 포트, 프로토콜(TCP 또는 UDP) 조합에 대해 다음 명령을 입력합니다.

   iptables -t nat -t PREROUTING -p <tcp|udp> -d <vip> --dport <port> -j REDIRECT

   이 명령을 실행하면 실제 서버가 제공된 VIP 및 포트로 향하는 패킷을 처리합니다.

2. 각 실제 서버에 구성을 저장합니다.

   ```
   # iptables-save > /etc/sysconfig/iptables
   # systemctl enable iptables.service
   ```

   systemctl enable 명령을 사용하면 네트워크가 시작되기 전에 시스템이 부팅 시 iptables 구성을 다시 로드합니다.

3.2.4. sysctl을 사용한 직접 라우팅

Direct Routing을 사용할 때 ARP 제한을 처리하는 또 다른 방법은 **sysctl** 인터페이스를 사용하는 것입니다. 관리자는 실제 서버가 ARP 요청에서 VIP를 알리지 않고 VIP 주소에 대한 ARP 요청에 응답하지 않도록 두 개의 **systcl** 설정을 구성할 수 있습니다. 이를 활성화하려면 다음 명령을 입력합니다.

```
echo 1 > /proc/sys/net/ipv4/conf/eth0/arp_ignore
echo 2 > /proc/sys/net/ipv4/conf/eth0/arp_announce
```

또는 **/etc/sysctl.d/arp.conf** 파일에 다음 행을 추가할 수도 있습니다.

```
net.ipv4.conf.eth0.arp_ignore = 1
net.ipv4.conf.eth0.arp_announce = 2
```

3.3. 구성을 함께 배치

사용할 이전 라우팅 방법 중 하나를 확인한 후 하드웨어를 함께 연결하고 구성해야 합니다.

중요

LVS 라우터의 네트워크 어댑터는 동일한 네트워크에 액세스하도록 구성해야 합니다. 예를 들어 **eth0** 이 공용 네트워크에 연결하고**eth1** 이 사설 네트워크에 연결된 경우 백업 LVS 라우터의 동일한 장치가 동일한 네트워크에 연결되어 있어야 합니다.

또한 부팅 시 첫 번째 인터페이스에 나열된 게이트웨이가 라우팅 테이블에 추가되고 다른 인터페이스에 나열된 후속 게이트웨이는 무시됩니다. 이는 실제 서버를 구성할 때 고려해야 할 특히 중요합니다.

하드웨어를 네트워크에 연결한 후 기본 및 백업 LVS 라우터에서 네트워크 인터페이스를 구성합니다. 이 작업은 네트워크 구성 파일을 수동으로 편집하여 수행해야 합니다. 네트워크 구성 파일 작업에 대한 자세한 내용은 Red Hat Enterprise Linux 7 네트워킹 가이드를 참조하십시오.

3.3.1. 일반 로드 밸런서 네트워킹 팁

Keepalived를 사용하여 로드 밸런서를 설정하기 전에 LVS 라우터의 공용 네트워크와 사설 네트워크의 실제 IP 주소를 구성합니다. 각 토폴로지의 섹션에서는 예제 네트워크 주소를 제공하지만 실제 네트워크 주소가 필요합니다. 다음은 네트워크 인터페이스를 시작하거나 상태를 확인하는 데 유용한 명령입니다.

실제 네트워크 인터페이스 가져오기

실제 네트워크 인터페이스를 열려면 **root** 로 다음 명령을 사용하여*N* 을 인터페이스(**eth0** 및 **eth1**)로 바꿉니다.

ifup eth*N*

주의

Keepalived(**eth0:1** 또는 **eth1:1**)를 사용하여 구성할 수 있는 유동 IP 주소를 여는 데 **ifup** 스크립트를 *사용하지* 마십시오. 대신 **service** 또는 **systemctl** 명령을 사용하여 **keepalived** 를 시작합니다.

실제 네트워크 인터페이스 가져오기

실제 네트워크 인터페이스를 중단하려면 **root** 로 다음 명령을 사용하여*N* 을 인터페이스(**eth0** 및 **eth1**)로 바꿉니다.

ifdown eth*N*

네트워크 인터페이스 상태 확인

지정된 시간에 어떤 네트워크 인터페이스가 있는지 확인해야 하는 경우 다음 명령을 입력합니다.

IP 링크

머신의 라우팅 테이블을 보려면 다음 명령을 실행합니다.

IP 경로

3.3.2. 방화벽 요구 사항

방화벽(**firewalld** 또는 **iptables** 를 통해)을 실행하는 경우 VRRP 트래픽이 **keepalived** 노드 간에 전달되도록 허용해야 합니다. **firewalld** 를 사용하여 VRRP 트래픽을 허용하도록 방화벽을 구성하려면 다음 명령을 실행합니다.

```
# firewall-cmd --add-rich-rule='rule protocol value="vrrp" accept' --permanent
# firewall-cmd --reload
```

영역이 생략되면 기본 영역이 사용됩니다.

그러나 **iptables** 를 사용하여 VRRP 트래픽을 허용해야 하는 경우 다음 명령을 실행합니다.

```
# iptables -I INPUT -p vrrp -j ACCEPT
# iptables-save > /etc/sysconfig/iptables
# systemctl restart iptables
```

3.4. 멀티 포트 서비스 및 로드 밸런서

모든 토폴로지 아래의 LVS 라우터는 다중 포트 로드 밸런서 서비스를 생성할 때 추가 구성이 필요합니다. 다중 포트 서비스는 방화벽 마크를 사용하여 서로 다른 번들이지만 HTTP(포트 80) 및 HTTPS(포트 443)와 같은 관련 프로토콜 또는 Load Balancer가 FTP와 같은 실제 다중 포트 프로토콜과 함께 사용되는 경우 인위적으로 생성할 수 있습니다. 두 경우 모두 LVS 라우터는 방화벽 표시를 사용하여 다른 포트로 향하지만 동일한 방화벽 마크가 있는 패킷이 동일하게 처리되어야 함을 인식합니다. 또한 지속성 매개 변수에 의해 지정된 시간 내에 연결이 발생하는 한 방화벽은 클라이언트 시스템의 연결이 동일한 호스트로 라우팅되도록 합니다.

IPVS는 실제 서버의 부하를 분산하는 데 사용되는 메커니즘을 통해 패킷에 할당된 방화벽 마크를 인식할 수 있지만 방화벽 표시는 자체적으로 할당할 수 없습니다. 방화벽 마크 *할당* 작업은 네트워크 패킷 필터인 **iptables** 에서 수행해야 합니다. Red Hat Enterprise Linux 7의 기본 방화벽 관리 툴은 **firewalld** 이며 **iptables** 를 구성하는 데 사용할 수 있습니다. 선호하는 경우 **iptables** 를 직접 사용할 수 있습니다. Red Hat Enterprise Linux 7의 iptables 사용에 대한 정보는 Red Hat Enterprise Linux 7 보안 가이드를 참조하십시오.

3.4.1. firewalld를 사용하여 방화벽 표시 할당

특정 포트를 대상으로 하는 패킷에 방화벽 표시를 할당하기 위해 관리자는 **firewalld** 의 **firewall-cmd** 유틸리티를 사용할 수 있습니다.

필요한 경우 **firewalld** 가 실행 중인지 확인합니다.

```
# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: active (running) since Tue 2016-01-26 05:23:53 EST; 7h ago
```

firewalld 를 시작하려면 다음을 입력합니다.

```
# systemctl start firewalld
```

시스템을 시작할 때 **firewalld** 가 활성화되어 있는지 확인하려면 다음을 수행하십시오.

```
# systemctl enable firewalld
```

이 섹션에서는 HTTP 및 HTTPS를 예로 번들하는 방법을 설명하지만 FTP는 다른 일반적으로 클러스터형 다중 포트 프로토콜입니다.

방화벽 마크를 사용할 때 유의해야 할 기본 규칙은 Keepalived에서 방화벽 표시를 사용하는 모든 프로토콜에 대해 네트워크 패킷에 마크를 할당하려면 영속 방화벽 규칙이 있어야 한다는 것입니다.

네트워크 패킷 필터 규칙을 만들기 전에 규칙이 없는지 확인합니다. 이렇게 하려면 쉘 프롬프트를 열고 **root** 로 로그인한 다음 다음 명령을 입력합니다.

```
# firewall-cmd --list-rich-rules
```

리치 규칙이 없으면 프롬프트가 즉시 다시 표시됩니다.

firewalld 가 활성 상태이고 리치 규칙이 있는 경우 규칙 세트가 표시됩니다.

규칙이 이미 있는 경우 **/etc/firewalld/zones/** 의 내용을 확인하고 진행하기 전에 안전한 장소에 보관할 수 있는 규칙을 복사합니다. 다음 형식으로 명령을 사용하여 원하지 않는 리치 규칙을 삭제합니다. **--permanent** 옵션은 설정이 지속되지만 명령은 다음 시스템 시작 시에만 적용됩니다.

```
firewall-cmd --zone=zone --remove-rich-rule='rule' --permanent
```

설정을 즉시 적용하는 데 필요한 경우 **--permanent** 옵션을 생략하는 명령을 반복합니다.

구성할 첫 번째 로드 밸런서 관련 방화벽 규칙은 Keepalived 서비스의 VRRP 트래픽이 작동하도록 허용하는 것입니다. 다음 명령을 실행합니다.

```
# firewall-cmd --add-rich-rule='rule protocol value="vrrp" accept' --permanent
```

영역이 생략되면 기본 영역이 사용됩니다.

다음은 포트 **80** 및 443에서 유동 IP 주소*n.n.n.n*.n 으로 향하는 들어오는 트래픽에 동일한 방화벽 마크 80을 할당하는 규칙입니다.

```
# firewall-cmd --add-rich-rule='rule family="ipv4" destination address="n.n.n.n/32" port port="80"
protocol="tcp" mark set="80"' --permanent
# firewall-cmd --add-rich-rule='rule family="ipv4" destination address="n.n.n.n/32" port port="443"
protocol="tcp" mark set="80"' --permanent
# firewall-cmd --reload
success
# firewall-cmd --list-rich-rules
rule protocol value="vrrp" accept
rule family="ipv4" destination address="n.n.n.n/32" port port="80" protocol="tcp" mark set=80
rule family="ipv4" destination address="n.n.n.n/32" port port="443" protocol="tcp" mark
set=80
```

영역이 생략되면 기본 영역이 사용됩니다.

firewalld 의 풍부한 언어 명령 사용에 대한 자세한 내용은Red Hat Enterprise Linux 7 보안 가이드를참조하십시오.

3.4.2. iptables를 사용하여 방화벽 마크 할당

방화벽 표시를 특정 포트로 향하는 패킷에 할당하기 위해 관리자는 **iptables** 를 사용할 수 있습니다.

이 섹션에서는 HTTP 및 HTTPS를 예로 번들하는 방법을 설명하지만 FTP는 다른 일반적으로 클러스터형 다중 포트 프로토콜입니다.

방화벽 마크를 사용할 때 유의해야 할 기본 규칙은 Keepalived에서 방화벽 표시를 사용하는 모든 프로토콜에 대해 네트워크 패킷에 마크를 할당하려면 영속 방화벽 규칙이 있어야 한다는 것입니다.

네트워크 패킷 필터 규칙을 만들기 전에 규칙이 없는지 확인합니다. 이렇게 하려면 쉘 프롬프트를 열고 **root** 로 로그인한 다음 다음 명령을 입력합니다.

/usr/sbin/service iptables status

iptables 가 실행되고 있지 않으면 프롬프트가 즉시 다시 표시됩니다.

iptables 가 활성화된 경우 규칙 세트가 표시됩니다. 규칙이 있는 경우 다음 명령을 입력합니다.

/sbin/service iptables stop

규칙이 이미 있는 경우 **/etc/sysconfig/iptables** 의 내용을 확인하고 진행하기 전에 안전한 장소에 보관해야 하는 규칙을 복사합니다.

방화벽 규칙과 관련된 첫 번째 로드 밸런서는 Keepalived 서비스가 작동할 수 있도록 VRRP 트래픽을 허용하는 것입니다.

```
/usr/sbin/iptables -I INPUT -p vrrp -j ACCEPT
```

다음은 포트 **80** 및 443에서 유동 IP 주소*n.n.n.n*.n 으로 향하는 들어오는 트래픽에 동일한 방화벽 마크 80을 할당하는 규칙입니다.

```
/usr/sbin/iptables -t mangle -A PREROUTING -p tcp -d n.n.n.n/32 -m multiport --dports 80,443 -
j MARK --set-mark 80
```

root 로 로그인하고 처음으로 규칙을 발행하기 전에**iptables** 에 대한 모듈을 로드해야 합니다.

위의 **iptables** 명령에서 *n.n.n.n*.n.n.n.n.n.n.n은 HTTP 및 HTTPS 가상 서버의 유동 IP로 교체해야 합니다. 이러한 명령은 적절한 포트에서 VIP로 주소가 지정된 모든 트래픽을 80의 방화벽 표시로 할당하여 IPVS에 의해 인식되고 적절하게 전달됩니다.

주의

위의 명령은 즉시 적용되지만 시스템을 재부팅하면 유지되지 않습니다.

3.5. FTP 구성

FTP(File Transport Protocol)는 로드 밸런서 환경에 고유한 일련의 문제를 제공하는 오래되고 복잡한 다중 포트 프로토콜입니다. 이러한 문제의 특성을 이해하려면 먼저 FTP의 작동 방식에 대한 몇 가지 핵심 사항을 이해해야 합니다.

3.5.1. FTP 작동 방식

대부분의 다른 서버 클라이언트 관계를 사용하면 클라이언트 시스템이 특정 포트에서 서버에 대한 연결을

열고 서버는 해당 포트에서 클라이언트에 응답합니다. FTP 클라이언트가 FTP 서버에 연결하면 FTP 제어 포트 21에 대한 연결이 열립니다. 그런 다음 *클라이언트*는 FTP *서버에 활성* 또는 *수동* 연결을 설정할지 여부를 알려줍니다. 클라이언트에서 선택한 연결 유형에 따라 서버가 응답하는 방법과 어떤 포트 트랜잭션이 발생하는지 결정합니다.

두 가지 유형의 데이터 연결은 다음과 같습니다.

활성 연결

활성 연결이 설정되면 *서버*는 포트 20에서 클라이언트 시스템의 높은 범위 포트로 클라이언트에 대한 데이터 연결을 엽니다. 그런 다음 서버의 모든 데이터가 이 연결을 통해 전달됩니다.

수동 연결

수동 연결이 설정되면 *클라이언트*는 FTP 서버에 10,000보다 높은 모든 포트에 있을 수 있는 수동 연결 포트를 설정하도록 요청합니다. 그런 다음 서버는 이 특정 세션에 대해 이 숫자가 높은 포트에 바인딩되고 해당 포트 번호를 클라이언트로 다시 중계합니다. 그런 다음 클라이언트는 데이터 연결에 대한 새로 바인딩된 포트를 엽니다. 클라이언트의 각 데이터 요청으로 인해 별도의 데이터 연결이 생성됩니다. 대부분의 최신 FTP 클라이언트는 서버에서 데이터를 요청할 때 수동 연결을 설정하려고 합니다.

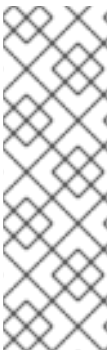
참고

*클라이언트*는 서버가 아닌 연결 유형을 결정합니다. 즉, FTP를 효과적으로 클러스터하려면 활성 연결 및 수동 연결을 모두 처리하도록 LVS 라우터를 구성해야 합니다.

FTP 클라이언트-서버 관계는 Keepalived에서 모르는 다수의 포트를 잠재적으로 열 수 있습니다.

3.5.2. 로드 밸런서 라우팅에 미치는 영향

IPVS 패킷 전달은 포트 번호 또는 방화벽 마크를 인식하는 것을 기반으로 클러스터 내외 연결만 허용합니다. 클러스터 외부에서 클라이언트가 포트 IPVS를 열려고 시도하지 않으면 연결이 삭제됩니다. 마찬가지로 실제 서버가 포트 IPVS에서 인터넷에 다시 연결을 열려고 하면 연결이 끊어집니다. 즉, 인터넷상의 FTP 클라이언트의 *모든* 연결에 동일한 방화벽 마크가 할당*되어야 하며* FTP 서버의 모든 연결이 네트워크 패킷 필터링 규칙을 사용하여 인터넷에 올바르게 전달 *되어야 합니다*.



참고

Passive FTP 연결을 활성화하려면 **ip_vs_ftp** 커널 모듈이 로드되어야 합니다. 쉘 프롬프트에서 다음 명령을 관리 사용자로 실행하여 이 모듈을 로드하고 모듈이 재부팅 시 로드되는지 확인합니다.

```
echo "ip_vs_ftp" >> /etc/modules-load.d/ip_vs_ftp.conf
systemctl enable systemd-modules-load
systemctl start systemd-modules-load
```

3.5.3. 네트워크 패킷 필터 규칙 생성

FTP 서비스에 **iptables** 규칙을 할당하기 전에 3.4절. "멀티 포트 서비스 및 로드 밸런서 "의 정보를 검토하여 기존 네트워크 패킷 필터링 규칙을 확인하기 위한 다중 포트 서비스 및 기술과 관련된 정보를 검토하십시오.

다음은 동일한 방화벽 마크 **21** 을 FTP 트래픽에 할당하는 규칙입니다.

3.5.3.1. 활성 연결에 대한 규칙

활성 연결 규칙은 커널에 포트 **20** (FTP 데이터 포트)의 *내부* 유동 IP 주소로 연결을 수락하고 전달하도록 지시합니다.

다음 **iptables** 명령을 사용하면 LVS 라우터에서 IPVS에서 알 수 없는 실제 서버의 발신 연결을 허용할 수 있습니다.

/usr/sbin/iptables -t nat -A POSTROUTING -p tcp -s *n.n.n*.0/24 --sport 20 -j MASQUERADE

iptables 명령에서 *n.n.n*.n은 **keepalived.conf** 파일의 NAT 인터페이스의 내부 네트워크 인터페이스 ***virtual_server*** 섹션에 대해 유동 IP의 처음 세 값으로 교체해야 합니다.

3.5.3.2. Passive Connections에 대한 규칙

수동 연결 규칙은 인터넷에서 들어오는 적절한 방화벽 표시를 다양한 포트 범위에서 서비스의 유동 IP 주소로 할당합니다. 10,000에서 20,000까지입니다.

> **주의**
>
> 수동 연결의 포트 범위를 제한하는 경우 일치하는 포트 범위를 사용하도록 FTP 서버 **vsftpd** 도 구성해야 합니다. 이 작업은**/etc/vsftpd.conf** 에 다음 행을 추가하여 수행할 수 있습니다.
>
> **pasv_min_port=10000**
>
> **pasv_max_port=20000**
>
> 실제 FTP 서버 주소를 덮어쓰도록 **pasv_address** 를 설정하면 LVS에서 가상 IP 주소로 업데이트되므로 사용해서는 안 됩니다.
>
> 다른 FTP 서버의 구성은 해당 문서를 참조하십시오.

이 범위는 대부분의 경우 충분히 넓어야 합니다. 그러나 아래 명령에서 **10000:20000** 를 **1024:65535** 로 변경하여 사용 가능한 모든 비보안 포트를 포함하도록 이 수를 늘릴 수 있습니다.

다음 **iptables** 명령은 적절한 포트에서 유동 IP로 주소가 지정된 모든 트래픽을**21** 의 방화벽 마크로 할당하는 효과가 있으며, 이는 IPVS로 인식되고 적절하게 전달됩니다.

/usr/sbin/iptables -t mangle -A PREROUTING -p tcp -d *n.n.n.n*/32 --dport 21 -j MARK --set-mark 21

/usr/sbin/iptables -t mangle -A PREROUTING -p tcp -d *n.n.n.n*/32 --dport 10000:20000 -j MARK --set-mark 21

iptables 명령에서 *n.n.n.n*.n.n.n.n.n.n.n.n.n.n.n.n.n.n.n.n.n.n.n.n은 **keepalived.conf** 파일의 ***virtual_server*** 섹션에 정의된 FTP 가상 서버의 유동 IP로 교체해야 합니다.

위의 명령은 즉시 적용되지만, 저장하지 않는 한 시스템을 재부팅하면 유지되지 않습니다. 변경 사항을 저장하려면 다음 명령을 입력합니다.

```
# iptables-save > /etc/sysconfig/iptables
```

시스템 시작 시 **iptables** 서비스가 시작되었는지 확인하려면 다음 명령을 입력합니다.

```
# systemctl enable iptables
```

다음 명령을 실행하고 변경 사항이 남아 있는지 확인하여 재부팅 시 변경 사항이 지속되는지 확인할 수 있습니다.

```
# systemctl restart iptables
```

3.6. 네트워크 패킷 필터 설정 저장

상황에 맞게 적절한 네트워크 패킷 필터를 구성한 후 재부팅 후 복원할 수 있도록 설정을 저장합니다. **iptables** 의 경우 다음 명령을 입력합니다.

```
# iptables-save > /etc/sysconfig/iptables
```

시스템 시작 시 **iptables** 서비스가 시작되었는지 확인하려면 다음 명령을 입력합니다.

```
# systemctl enable iptables
```

다음 명령을 실행하고 변경 사항이 남아 있는지 확인하여 재부팅 시 변경 사항이 지속되는지 확인할 수 있습니다.

```
# systemctl restart iptables
```

Red Hat Enterprise Linux 7에서iptables 사용에 대한 자세한 내용은 Red Hat Enterprise Linux 7 보안 가이드를 참조하십시오.

3.7. 패킷 전달 및 비로컬 바인딩 설정

Keepalived 서비스가 네트워크 패킷을 실제 서버로 올바르게 전달하려면 각 라우터 노드에 커널에 IP 전달이 설정되어 있어야 합니다. **root** 로 로그인하고 **/etc/sysctl.conf** 에서 **net.ipv4.ip_forward = 0** 를 읽는 행을 다음으로 변경합니다.

```
net.ipv4.ip_forward = 1
```

시스템을 재부팅할 때 변경 사항이 적용됩니다.

HAProxy 및 Keepalived의 로드 밸런싱도 동시에 *비로컬* 인 IP 주소에 바인딩할 수 있어야 합니다. 즉, 로컬 시스템의 장치에 할당되지 않습니다. 이렇게 하면 실행 중인 로드 밸런서 인스턴스가 페일오버를 위해 로컬이 아닌 IP에 바인딩할 수 있습니다.

활성화하려면 **/etc/sysctl.conf** 에서 **net.ipv4.ip_nonlocal_bind** 을 다음과 같은 행을 편집합니다.

```
net.ipv4.ip_nonlocal_bind = 1
```

시스템을 재부팅할 때 변경 사항이 적용됩니다.

IP 전달이 설정되어 있는지 확인하려면 **root** 로 다음 명령을 실행합니다.

/usr/sbin/sysctl net.ipv4.ip_forward

로컬 바인딩이 설정되어 있는지 확인하려면 **root** 로 다음 명령을 실행합니다.

/usr/sbin/sysctl net.ipv4.ip_nonlocal_bind

위의 두 명령에서 모두 **1** 를 반환하면 해당 설정이 활성화됩니다.

3.8. 실제 서버에서 서비스 구성

실제 서버가 Red Hat Enterprise Linux 시스템인 경우 부팅 시 활성화되도록 적절한 서버 데몬을 설정합니다. 이러한 데몬에는 웹 서비스용 **httpd** 또는 FTP 또는 Telnet 서비스의 경우**xinetd** 가 포함될 수 있습니다.

또한 실제 서버에 원격으로 액세스하는 것이 유용할 수 있으므로 **sshd** 데몬도 설치되어 실행 중이어야 합니다.

4장. KEEPALIVED를 사용한 초기 로드 밸런서 구성

Load Balancer 패키지를 설치한 후 Keepalived에 사용할 LVS 라우터 및 실제 서버를 설정하려면 몇 가지 기본 단계를 수행해야 합니다. 이 장에서는 이러한 초기 단계를 자세히 설명합니다.

4.1. 기본 KEEPALIVED 구성

이 기본 예에서는 두 개의 시스템이 로드 밸런서로 구성됩니다. LB1(Active) 및 LB2(Backup)는 실제 IP 주소 192.168.1.20에서 192.168.1.24로 번호가 지정된 **httpd** 웹 서버 풀에 대한 라우팅 요청으로, 가상 IP 주소를 10.0.0.1로 공유합니다. 각 로드 밸런서에는 두 개의 인터페이스(**eth0** 및 **eth1**)가 있으며, 하나는 외부 인터넷 트래픽을 처리하는 데 사용되며 다른 하나는 실제 서버로 요청을 라우팅하는 데 사용됩니다. 사용된 로드 밸런싱 알고리즘은 Round Robin이며 라우팅 방법은 Network Address Translation입니다.

4.1.1. keapalived.conf 파일 생성

keepalived는 로드 밸런서로 구성된 각 시스템의 **keepalived.conf** 파일을 통해 구성됩니다. 4.1절. "기본 Keepalived 구성" 예제와 같은 로드 밸런서 토폴로지를 생성하려면 텍스트 편집기를 사용하여 활성 및 백업 로드 밸런서, LB1 및 LB2 모두에서 **keepalived.conf** 를 엽니다. 예를 들면 다음과 같습니다.

```
vi /etc/keepalived/keepalived.conf
```

4.1절. "기본 Keepalived 구성" 에 설명된 대로 구성이 포함된 기본 로드 밸런싱 시스템에는 다음 코드 섹션에 설명된 대로 **keepalived.conf** 파일이 있습니다. 이 예에서 **keepalived.conf** 파일은 VRRP 인스턴스를 제외하고 활성 및 백업 라우터 모두에서 동일합니다. 4.1.1.2절. "VRRP 인스턴스"

4.1.1.1. 글로벌 정의

keepalived.conf 파일의 글로벌 정의 섹션을 사용하면 관리자가 로드 밸런서를 변경할 때 알림 세부 정보를 지정할 수 있습니다. 글로벌 정의는 선택 사항이며 Keepalived 구성에 필요하지 않습니다. **keepalived.conf** 파일의 이 섹션은 LB1 및 LB2 모두에서 동일합니다.

```
global_defs {

  notification_email {
    admin@example.com
  }
  notification_email_from noreply@example.com
  smtp_server 127.0.0.1
  smtp_connect_timeout 60
}
```

notification_email 는 로드 밸런서의 관리자이며, ***notification_email_from*** 는 로드 밸런서 상태가 변경되는 주소입니다. SMTP별 구성은 알림이 발송되는 메일 서버를 지정합니다.

4.1.1.2. VRRP 인스턴스

다음 예제에서는 마스터 라우터 및 백업 라우터에 **keeplalived.conf** 파일의 ***vrrp_sync_group*** 스탠자를 보여줍니다. **state** 및 **priority** 값은 두 시스템 간에 다릅니다.

다음 예제에서는 LB1의 **keepalived.conf** 파일에 대한 ***vrrp_sync_group*** 스탠자를 보여줍니다.

```
vrrp_sync_group VG1 {
  group {
```

```
    RH_EXT
    RH_INT
  }
}

vrrp_instance RH_EXT {
  state MASTER
  interface eth0
  virtual_router_id 50
  priority 100
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass passw123
  }
  virtual_ipaddress {
  10.0.0.1
  }
}

vrrp_instance RH_INT {
  state MASTER
  interface eth1
  virtual_router_id 2
  priority 100
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass passw123
  }
  virtual_ipaddress {
    192.168.1.1
  }
}
```

다음 예제에서는 LB2용 **keepalived.conf** 파일의 ***vrrp_sync_group*** 스탠자를 백업 라우터로 보여줍니다.

```
vrrp_sync_group VG1 {
  group {
    RH_EXT
    RH_INT
  }
}

vrrp_instance RH_EXT {
  state BACKUP
  interface eth0
  virtual_router_id 50
  priority 99
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass passw123
  }
  virtual_ipaddress {
  10.0.0.1
```

```
    }
}

vrrp_instance RH_INT {
  state BACKUP
  interface eth1
  virtual_router_id 2
  priority 99
  advert_int 1
  authentication {
     auth_type PASS
     auth_pass passw123
  }
  virtual_ipaddress {
     192.168.1.1
  }
}
```

이 예에서 ***vrrp_sync_group*** 스탠자는 모든 상태 변경(예: 페일오버)을 통해 함께 유지되는 VRRP 그룹을 정의합니다. 인터넷(RH_EXT)과 통신하는 외부 인터페이스와 내부 인터페이스(RH_INT)용 인스턴스가 정의되어 있습니다.

vrrp_instance 행은 가상 IP 인스턴스를 생성하는 VRRP 서비스 데몬의 가상 인터페이스 구성을 자세히 설명합니다. **MASTER** 는 활성 서버를 지정하고상태는 백업 서버를 지정합니다.

interface 매개 변수는 이 특정 가상 IP 인스턴스에 물리적 인터페이스 이름을 할당합니다.

virtual_router_id 는 가상 라우터 인스턴스의 숫자 식별자입니다. 이 가상 라우터에 참여하는 모든 LVS 라우터 시스템에서 동일해야 합니다. 동일한 네트워크 인터페이스에서 **keepalived** 의 여러 인스턴스를 구별하는 데 사용됩니다.

우선순위 는 할당된 인터페이스가 장애 조치(failover)에서 인수되는 순서를 지정합니다. 즉 숫자가 클수록 우선순위가 높습니다. 이 우선순위 값은 0에서 255 사이의 범위 내에 있어야 하며, **MASTER** 상태로 구성된 **Load Balancing** 서버에는 **state** 로 구성된 서버의 우선순위 값보다 높은 우선순위 값이 설정되어 있어야 합니다.

인증 블록은 장애 조치(failover) 동기화를 위해 서버를 인증하는 데 사용되는 인증 유형(**auth_type**) 및 암호(**auth_pass**)를 지정합니다. **PASS** 는 암호 인증을 지정합니다. Keepalived는**AH** 또는 연결 무결성을 위해 인증 헤더도 지원합니다.

마지막으로 **virtual_ipaddress** 옵션은 인터페이스 가상 IP 주소를 지정합니다.

4.1.1.3. 가상 서버 정의

keepalived.conf 파일의 가상 서버 정의 섹션은 LB1 및 LB2 모두에서 동일합니다.

```
virtual_server 10.0.0.1 80 {
   delay_loop 6
   lb_algo rr
   lb_kind NAT
   protocol TCP

   real_server 192.168.1.20 80 {
      TCP_CHECK {
            connect_timeout 10
      }
```

```
    }
    real_server 192.168.1.21 80 {
        TCP_CHECK {
            connect_timeout 10
        }
    }
    real_server 192.168.1.22 80 {
        TCP_CHECK {
            connect_timeout 10
        }
    }
    real_server 192.168.1.23 80 {
        TCP_CHECK {
            connect_timeout 10
        }
    }

}
```

이 블록에서는 **virtual_server** 가 먼저 IP 주소로 구성됩니다. 그런 다음**delay_loop** 는 상태 점검 간 시간(초)을 구성합니다. **lb_algo** 옵션은 가용성에 사용되는 알고리즘의 종류를 지정합니다(이 경우 Round-Robin의 경우 **rr**; 가능한 **lb_algo** 값 목록은 표 4.1. "lv_algo values for Virtual Server"참조). **lb_kind** 옵션은 라우팅 방법을 결정합니다. 이 경우 네트워크 주소 변환(또는 **nat**)이 사용됩니다.

가상 서버 세부 정보를 구성한 후 먼저 IP 주소를 지정하여 **real_server** 옵션이 다시 구성됩니다. **TCP_CHECK** 스탠자는 TCP를 사용하여 실제 서버의 가용성을 확인합니다.**connect_timeout** 은 시간 초과가 발생하기 전의 시간(초)을 구성합니다.

참고

로드 밸런서 또는 실제 서버 중 하나에서 가상 IP에 액세스하는 것은 지원되지 않습니다. 마찬가지로 실제 서버와 동일한 시스템에서 로드 밸런서를 구성하는 것은 지원되지 않습니다.

표 4.1. lv_algo values for Virtual Server

알고리즘 이름	lv_algo 값
round-Robin	**rr**
weighted Round-Robin	**wrr**
least-Connection	**lc**
가중치 Least-Connection	**wlc**
locality-Based Least-Connection	**lblc**
복제를 사용한 지역 기반 Least-Connection 스케줄링	**lblcr**
대상 해시	**dh**
소스 해시	**sh**

알고리즘 이름	lv_algo 값
소스 Expected Delay	**sed**
never Queue	**nq**

4.2. KEEPALIVED DIRECT ROUTING 구성

Keepalived의 직접 라우팅 구성은 NAT와의 구성과 유사합니다. 다음 예에서 Keepalived는 포트 80에서 HTTP를 실행하는 실제 서버 그룹에 대한 부하 분산을 제공하도록 구성되어 있습니다. 직접 라우팅을 구성하려면 ***lb_kind*** 매개변수를 **DR** 로 변경합니다. 기타 구성 옵션은 4.1절. "기본 Keepalived 구성" 에서 설명합니다.

다음 예제에서는 직접 라우팅을 사용하는 Keepalived 구성에서 활성 서버에 대한 **keepalived.conf** 파일을 보여줍니다.

```
global_defs {
   notification_email {
     admin@example.com
   }
   notification_email_from noreply_admin@example.com
   smtp_server 127.0.0.1
   smtp_connect_timeout 60
}

vrrp_instance RH_1 {
    state MASTER
    interface eth0
    virtual_router_id 50
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass passw123
    }
    virtual_ipaddress {
        172.31.0.1
    }
}

virtual_server 172.31.0.1 80 {
    delay_loop 10
    lb_algo rr
    lb_kind DR
    persistence_timeout 9600
    protocol TCP

    real_server 192.168.0.1 80 {
        weight 1
        TCP_CHECK {
          connect_timeout 10
          connect_port    80
        }
    }
```

```
    real_server 192.168.0.2 80 {
        weight 1
        TCP_CHECK {
          connect_timeout 10
          connect_port    80
        }
    }
    real_server 192.168.0.3 80 {
        weight 1
        TCP_CHECK {
          connect_timeout 10
          connect_port    80
        }
    }
}
```

다음 예제에서는 직접 라우팅을 사용하는 Keepalived 구성에서 백업 서버에 대한 **keepalived.conf** 파일을 보여줍니다. **state** 및 **priority** 값은 활성 서버의 **keepalived.conf** 파일과 다릅니다.

```
global_defs {
   notification_email {
     admin@example.com
   }
   notification_email_from noreply_admin@example.com
   smtp_server 127.0.0.1
   smtp_connect_timeout 60
}

vrrp_instance RH_1 {
    state BACKUP
    interface eth0
    virtual_router_id 50
    priority 99
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass passw123
    }
    virtual_ipaddress {
        172.31.0.1
    }
}

virtual_server 172.31.0.1 80 {
    delay_loop 10
    lb_algo rr
    lb_kind DR
    persistence_timeout 9600
    protocol TCP

    real_server 192.168.0.1 80 {
        weight 1
        TCP_CHECK {
          connect_timeout 10
          connect_port    80
        }
```

```
    }
    real_server 192.168.0.2 80 {
        weight 1
        TCP_CHECK {
          connect_timeout 10
          connect_port    80
        }
    }
    real_server 192.168.0.3 80 {
        weight 1
        TCP_CHECK {
          connect_timeout 10
          connect_port    80
        }
    }
}
```

4.3. 서비스 시작

로드 밸런서 구성의 서버에서 다음 명령을 입력하여 서비스를 시작합니다.

```
# systemctl start keepalived.service
```

재부팅을 통해 Keepalived 서비스를 유지하려면 로드 밸런서 구성의 서버에 다음 명령을 입력합니다.

```
# systemctl enable keepalived.service
```

5장. HAPROXY 설정

이 장에서는 고가용성 환경에 HAProxy 서비스를 배포할 때 관리자가 직면할 수 있는 일반적인 구성 옵션을 강조하는 기본 설정 구성에 대해 설명합니다.

HAProxy에는 로드 밸런싱을 위한 자체 스케줄링 알고리즘 세트가 있습니다. 이러한 알고리즘은 5.1절. "HAProxy 스케줄링 알고리즘" 에 설명되어 있습니다.

HAProxy는 **/etc/haproxy/haproxy.cfg** 파일을 편집하여 구성됩니다.

HAProxy를 사용한 로드 밸런서 구성은 구성에 대한 다음 5개의 섹션으로 구성됩니다.

- 5.2절. "글로벌 설정"
- 4개의 하위 섹션으로 구성된 proxies 섹션:
  - 5.3절. "기본 설정" 설정
  - 5.4절. "프런트 엔드 설정" 설정
  - 5.5절. "백엔드 설정" 설정

5.1. HAPROXY 스케줄링 알고리즘

로드 밸런싱에 대한 HAProxy 예약 알고리즘은 **/etc/haproxy/haproxy.cfg** 구성 파일의 ***backend*** 섹션에 있는 ***balance*** 매개변수에서 편집할 수 있습니다. HAProxy는 여러 백엔드가 있는 구성을 지원하며 각 백엔드는 스케줄링 알고리즘을 사용하여 구성할 수 있습니다.

라운드 로빈 (***roundrobin***)

실제 서버 풀 주위에 각 요청을 순차적으로 배포합니다. 이 알고리즘을 사용하면 모든 실제 서버가 용량 또는 로드와 관계없이 동일하게 처리됩니다. 이 스케줄링 모델은 라운드 로빈 DNS와 유사하지만 호스트 기반이 아닌 네트워크 연결 기반이므로 더 세분화됩니다. 로드 밸런서 라운드 로빈 스케줄링은 캐시된 DNS 쿼리로 인한 불균형이 발생하지 않습니다. 그러나 HAProxy에서는 이 스케줄러를 사용하여 서버 가중치의 구성을 수행할 수 있으므로 활성 서버 수는 백엔드당 4095로 제한됩니다.

정적 라운드 로빈 (***static-rr***)

각 요청을 실제 서버 풀에 순차적으로 배포하지만 *Round-Robin* 에서는 서버 가중치 구성을 동적으로 허용하지 않습니다. 그러나 서버 가중치의 정적 특성으로 인해 백엔드의 활성 서버 수에 제한이 없습니다.

최소 연결 (***leastconn***)

활성 연결이 더 적은 실제 서버에 더 많은 요청을 배포합니다. 세션 또는 연결 길이가 다양한 동적 환경을 사용하는 관리자는 이 스케줄러가 환경에 더 적합할 수 있습니다. 또한 관리자가 이 스케줄러를 사용하여 이동 중에 가중치를 조정할 수 있으므로 서버 그룹이 다른 용량을 가진 환경에도 이상적입니다.

소스 (***source***)

소스 IP 주소를 요청하고 실행 중인 모든 서버의 가중치로 나뉘어 요청을 받을 서버를 결정하여 서버에 요청을 배포합니다. 모든 서버가 실행 중인 시나리오에서는 소스 IP 요청이 동일한 실제 서버에서 일관되게 제공됩니다. 실행 중인 서버의 수 또는 가중치가 변경되면 해시/체중 결과가 변경되었기 때문에 세션이 다른 서버로 이동될 수 있습니다.

URI (***uri***)

전체 URI(또는 URI의 구성 가능한 부분)를 해시하여 서버에 요청을 배포하고, 실행 중인 모든 서버의 가중치를 지정하여 요청할 서버를 결정합니다. 모든 활성 서버가 실행 중인 시나리오에서는 대상 IP 요청이

동일한 실제 서버에서 일관되게 제공됩니다. 이 스케줄러는 URI의 디렉토리 부분 시작 시 문자 길이와 해시 결과를 계산하기 위해 URI의 슬래시로 지정된 URI의 디렉터리 깊이로 추가로 구성할 수 있습니다.

URL 매개변수 (***url_param***)

소스 URL 요청에서 특정 매개변수 문자열을 찾고 실행 중인 모든 서버의 가중치로 해시 계산을 수행하여 서버에 요청을 배포합니다. URL에서 매개변수가 없는 경우 스케줄러는 기본적으로 Round-robin 스케줄링으로 설정됩니다. 수정자는 POST 매개변수를 기반으로 사용할 수 있으며, 관리자가 해시 결과를 계산하기 전에 관리자가 특정 매개 변수의 weight에 할당하는 최대 옥텟 수에 따라 대기 제한을 사용할 수 있습니다.

헤더 이름 (***hdr***)

각 소스 HTTP 요청에서 특정 헤더 이름을 확인하고 실행 중인 모든 서버의 가중치로 해시 계산을 수행하여 서버에 요청을 분산합니다. 헤더가 없는 경우 스케줄러는 기본적으로 Round-robin 스케줄링으로 설정됩니다.

RDP Cryostat (***rdp-cookie***)

모든 TCP 요청에 대해 RDP 쿠키를 검색하고 실행 중인 모든 서버의 가중치로 나눈 해시 계산을 수행하여 서버에 요청을 배포합니다. 헤더가 없는 경우 스케줄러는 기본적으로 Round-robin 스케줄링으로 설정됩니다. 이 방법은 세션 무결성을 유지 관리할 때 지속성에 이상적입니다.

5.2. 글로벌 설정

global 설정은 HAProxy를 실행하는 모든 서버에 적용되는 매개변수를 구성합니다. 일반적인 ***global*** 섹션은 다음과 같을 수 있습니다.

```
global
    log 127.0.0.1 local2
    maxconn 4000
    user haproxy
    group haproxy
    daemon
```

위의 구성에서 관리자는 서비스를 로컬 **syslog** 서버에 대한 모든 항목을 ***log*** 로 구성했습니다. 기본적으로 이 위치는 **/var/log/syslog** 또는 일부 사용자 지정 위치일 수 있습니다.

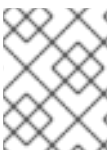
maxconn 매개변수는 서비스의 최대 동시 연결 수를 지정합니다. 기본적으로 최대값은 2000입니다.

user 및 ***group*** 매개변수는 **haproxy** 프로세스가 속한 사용자 이름과 그룹 이름을 지정합니다.

마지막으로 ***daemon*** 매개변수는 **haproxy** 가 백그라운드 프로세스로 실행되도록 지정합니다.

5.3. 기본 설정

default 설정은 구성(***frontend***, ***backend***, ***listen***)의 모든 프록시 하위 섹션에 적용되는 매개변수를 구성합니다. 일반적인 ***default*** 섹션은 다음과 같을 수 있습니다.



참고

proxy 하위 섹션(***frontend***, ***backend*** 또는 ***listen***)에 구성된 매개변수는 ***default*** 의 매개변수 값보다 우선합니다.

```
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
```

mode HAProxy 인스턴스의 프로토콜을 지정합니다. ***http*** 모드를 사용하면 로드 밸런싱 웹 서버에 이상적인 HTTP를 기반으로 소스 요청을 실제 서버에 연결합니다. 기타 애플리케이션의 경우 ***tcp*** 모드를 사용합니다.

log 로그 항목이 기록되는 로그 주소 및**syslog** 기능을 지정합니다. ***global*** 값은 HAProxy 인스턴스를 ***global*** 섹션의 ***log*** 매개변수에 지정된 것과 참조합니다.

option httplog HTTP 요청, 세션 상태, 연결 번호, 소스 주소 및 연결 타이머를 포함하여 HTTP 세션의 다양한 값을 로깅할 수 있습니다.

option dontlognull null 연결 로깅을 비활성화합니다. 즉, HAProxy는 데이터가 전송되지 않은 연결을 기록하지 않습니다. null 연결이 취약점 공개 포트 검사와 같은 악의적인 활동을 표시할 수 있는 인터넷상의 웹 애플리케이션과 같은 환경에는 권장되지 않습니다.

retries 첫 번째 시도에서 연결 실패 후 실제 서버가 연결 요청을 재시도하는 횟수를 지정합니다.

다양한 ***timeout*** 값은 지정된 요청, 연결 또는 응답의 비활성 시간을 지정합니다. 이러한 값은 일반적으로 밀리초 단위로 표시되지만 단위의 앞에 숫자 값의 접미사를 지정하여 다른 단위로 표현될 수 있습니다. 지원되는 단위는 당사(마이크로초), ms(밀리초), s(초), m(분), h(시간) 및 d(일)입니다. ***http-request 10s*** 클라이언트의 전체 HTTP 요청을 기다리는 데 10초를 지정합니다. ***queue 1m*** 연결이 삭제되기 전에 대기하는 시간으로 1분을 설정하고 클라이언트가 503 또는 "Service Unavailable" 오류를 수신합니다. ***connect 10s*** 서버에 성공적으로 연결할 때까지 대기하는 시간(초)을 지정합니다. ***client 1m*** 클라이언트가 비활성 상태로 유지할 수 있는 시간(분)을 지정합니다(사용하거나 데이터를 전송할 수 없음). ***server 1m*** 시간 초과가 발생하기 전에 서버가 데이터를 수락하거나 전송할 수 있는 시간(분)을 지정합니다.

5.4. 프런트 엔드 설정

frontend 설정은 클라이언트 연결 요청에 대해 서버의 수신 소켓을 구성합니다.***frontend*** 의 일반적인 HAProxy 구성은 다음과 같을 수 있습니다.

```
frontend  main
  bind 192.168.0.10:80
  default_backend app
```

main 라는 ***frontend*** 는 192.168.0.10 IP 주소로 구성되며 ***bind*** 매개변수를 사용하여 포트 80에서 수신 대기합니다. 연결된 ***use backend*** 는 모든 세션이 ***app*** 백엔드에 연결되도록 지정합니다.

5.5. 백엔드 설정

backend 설정은 로드 밸런서 스케줄링 알고리즘뿐만 아니라 실제 서버 IP 주소를 지정합니다. 다음 예제에서는 일반적인 ***backend*** 섹션을 보여줍니다.

-

```
backend app
  balance     roundrobin
  server  app1 192.168.1.1:80 check
  server  app2 192.168.1.2:80 check
  server  app3 192.168.1.3:80 check inter 2s rise 4 fall 3
  server  app4 192.168.1.4:80 backup
```

백엔드 서버의 이름은 ***app*** 입니다. ***balance*** 는 사용할 로드 밸런서 스케줄링 알고리즘을 지정합니다. 이 경우 Round Robin (***roundrobin***)이지만 HAProxy에서 지원하는 모든 스케줄러일 수 있습니다. HAProxy에서 스케줄러를 구성하는 방법에 대한 자세한 내용은 5.1절. "HAProxy 스케줄링 알고리즘" 을 참조하십시오.

server 행은 백엔드에서 사용 가능한 서버를 지정합니다.***app1 app4*** 는 각 실제 서버에 내부적으로 할당된 이름입니다. 로그 파일은 이름별로 서버 메시지를 지정합니다. 주소는 할당된 IP 주소입니다. IP 주소의 콜론 뒤의 값은 특정 서버에서 연결이 수행되는 포트 번호입니다. ***check*** 옵션은 주기적인 상태 점검을 위해 서버에 플래그를 지정하여 데이터를 수신 및 전송하고 세션 요청을 수행할 수 있도록 합니다. 또한 서버 app3에서는 상태 점검 간격을 2초(***inter 2s***)로 설정하고, 서버가 정상(***rise 4***)으로 간주되는지 확인하기 위해 전달해야 하며, 서버가 실패(***fall 3***)로 간주되기 전에 연속으로 확인에 실패하는 횟수().

5.6. HAPROXY 시작

HAProxy 서비스를 시작하려면 다음 명령을 입력합니다.

```
# systemctl start haproxy.service
```

재부팅을 통해 HAProxy 서비스를 영구적으로 설정하려면 다음 명령을 입력합니다.

```
# systemctl enable haproxy.service
```

5.7. RSYSLOG에 HAPROXY 메시지 로깅

/dev/log 소켓에 작성하여 **rsyslog** 에 HAProxy 메시지를 기록하도록 시스템을 구성할 수 있습니다. 또는 TCP 루프백 주소를 대상으로 지정할 수 있지만 이로 인해 성능이 느려집니다.

다음 절차에서는 **rsyslog** 에 메시지를 기록하도록 HAProxy를 구성합니다.

1. HAProxy 구성 파일의 **global** 섹션에서 **log** 지시문을 사용하여 **/dev/log** 소켓을 대상으로 합니다.

   ```
   log /dev/log local0
   ```

2. **frontend,backend** 및 **listen** 프록시를 업데이트하여 HAProxy 구성 파일의**global** 섹션에서 구성한 **rsyslog** 서비스에 메시지를 보냅니다. 이렇게 하려면 다음과 같이 구성 파일의**defaults** 섹션에 **log global** 지시문을 추가합니다.

   ```
   defaults
     log global
     option httplog
   ```

3. **chroot**된 환경에서 HAProxy를 실행 중이거나 HAProxy가 chroot 구성 지시문을 사용하여**chroot** 디렉토리를 생성하도록 하는 경우 해당 **chroot** 디렉터리 내에서 소켓을 사용할 수 있어야 합니다. 이렇게 하려면 **rsyslog** 구성을 수정하여 **chroot** 파일 시스템 내에 새 수신 대기 소켓을 생성할 수 있습니다. 이렇게 하려면 **rsyslog** 구성 파일에 다음 행을 추가합니다.

```
$ModLoad imuxsock
$AddUnixListenSocket PATH_TO_CHROOT/dev/log
```

4. HAProxy 로그 메시지가 표시되는 내용과 위치를 사용자 지정하려면 *시스템 관리자 가이드*의 Rsyslog 기본 구성에 설명된 대로 **rsyslog** 필터를 사용할 수 있습니다.

부록 A. 설정 예: HAPROXY 및 KEEPALIVED를 사용하여 CEPH OBJECT GATEWAY 서버 로드

이 부록에서는 Ceph 클러스터와 함께 HAProxy 및 Keepalived 구성을 보여주는 예를 제공합니다. Ceph Object Gateway를 사용하면 로드가 증가할 때 확장할 수 있도록 오브젝트 게이트웨이의 많은 인스턴스를 단일 영역에 할당할 수 있습니다. 각 오브젝트 게이트웨이 인스턴스에는 자체 IP 주소가 있으므로 HAProxy 및 keepalived를 사용하여 Ceph Object Gateway 서버 간에 부하를 분산할 수 있습니다.

이 구성에서 HAproxy는 Ceph Object Gateway 서버에서 로드 밸런싱을 수행하는 반면 Keepalived는 Ceph Object Gateway 서버의 가상 IP 주소를 관리하고 HAProxy를 모니터링하는 데 사용됩니다.

HAProxy 및 keepalived의 또 다른 사용 사례는 HAProxy 서버에서 HTTPS를 종료하는 것입니다. RHCS(Red Hat Ceph Storage) 1.3.x는 Civetweb을 사용하며 RHCS 1.3.x의 구현은 HTTPS를 지원하지 않습니다. HAProxy 서버를 사용하여 HAProxy 서버에서 HTTPS를 종료하고 HAProxy 서버와 Civetweb 게이트웨이 인스턴스 간에 HTTP를 사용할 수 있습니다. 이 예제에는 이 구성이 절차의 일부로 포함됩니다.

A.1. 사전 요구 사항

Ceph Object Gateway를 사용하여 HAProxy를 설정하려면 다음이 있어야 합니다.

- 실행 중인 Ceph 클러스터
- 포트 80에서 실행되도록 구성된 동일한 영역에 있는 두 개 이상의 Ceph Object Gateway 서버
- HAProxy 및 keepalived용 서버가 두 개 이상 있어야 합니다.

참고

이 절차에서는 Ceph Object Gateway 서버가 두 개 이상 실행되고 포트 80을 통해 테스트 스크립트를 실행할 때 유효한 응답을 가져오는 것으로 가정합니다.

A.2. HAPROXY 노드 준비

다음 설정에서는 **haproxy** 및 **haproxy2** 라는 두 개의 HAProxy 노드와**rgw1** 및 **rgw2** 라는 두 개의 Ceph Object Gateway 서버를 가정합니다. 원하는 이름 지정 규칙을 사용할 수 있습니다. 두 개의 HAProxy 노드에서 다음 절차를 수행합니다.

1. Install Red Hat Enterprise Linux 7.
2. 노드를 등록합니다.

```
# subscription-manager register
```

3. Red Hat Enterprise Linux 7 서버 리포지토리를 활성화합니다.

```
# subscription-manager repos --enable=rhel-7-server-rpms
```

4. 서버를 업데이트합니다.

```
# yum update -y
```

5. 필요에 따라 관리자 도구(예: **wget**,**vim** 등)를 설치합니다.

6. 포트 80을 엽니다.

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

7. HTTPS의 경우 포트 443을 엽니다.

```
# firewall-cmd --zone=public --add-port 443/tcp --permanent
# firewall-cmd --reload
```

A.3. KEEPALIVED 설치 및 구성

두 개의 HAProxy 노드에서 다음 절차를 수행합니다.

1. keepalived를 설치합니다.

```
# yum install -y keepalived
```

2. keepalived를 구성합니다.

```
# vim /etc/keepalived/keepalived.conf
```

다음 구성에는 HAProxy 프로세스를 확인하는 스크립트가 있습니다. 인스턴스는 **eth0** 을 네트워크 인터페이스로 사용하고 **haproxy** 를 마스터 서버로 구성하고**haproxy2** 를 백업 서버로 구성합니다. 또한 가상 IP 주소 192.168.0.100을 할당합니다.

```
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 VI_1 {
  interface eth0 # interface to monitor
  state MASTER # MASTER on haproxy, BACKUP on haproxy2
  virtual_router_id 51
  priority 101 # 101 on haproxy, 100 on haproxy2
  virtual_ipaddress {
    192.168.0.100 # virtual ip address
  }
  track_script {
    chk_haproxy
  }
}
```

3. keepalived를 활성화하고 시작합니다.

```
# systemctl enable keepalived
# systemctl start keepalived
```

A.4. HAPROXY 설치 및 구성

두 개의 HAProxy 노드에서 다음 절차를 수행합니다.

1. **haproxy** 를 설치합니다.

```
# yum install haproxy
```

2. SELinux 및 HTTP에 대한 **haproxy** 를 구성합니다.

```
# vim /etc/firewalld/services/haproxy-http.xml
```

다음 행을 추가합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<service>
<short>HAProxy-HTTP</short>
<description>HAProxy load-balancer</description>
<port protocol="tcp" port="80"/>
</service>
```

root 로서 **haproxy-http.xml** 파일에 올바른 SELinux 컨텍스트 및 파일 권한을 할당합니다.

```
# cd /etc/firewalld/services
# restorecon haproxy-http.xml
# chmod 640 haproxy-http.xml
```

3. HTTPS를 사용하려면 SELinux 및 HTTPS에 대해 **haproxy** 를 구성합니다.

```
# vim /etc/firewalld/services/haproxy-https.xml
```

다음 행을 추가합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<service>
<short>HAProxy-HTTPS</short>
<description>HAProxy load-balancer</description>
<port protocol="tcp" port="443"/>
</service>
```

root 로서 **haproxy-https.xml** 파일에 올바른 SELinux 컨텍스트 및 파일 권한을 할당합니다.

```
# cd /etc/firewalld/services
# restorecon haproxy-https.xml
# chmod 640 haproxy-https.xml
```

4. HTTPS를 사용하려면 SSL에 대한 키를 생성합니다. 인증서가 없는 경우 자체 서명 인증서를 사용할 수 있습니다. 키 및 자체 서명된 인증서를 생성하는 방법에 대한 자세한 내용은 Red Hat Enterprise Linux *시스템 관리자 가이드를 참조하십시오.*

마지막으로 인증서와 키를 PEM 파일에 넣습니다.

```
# cat example.com.crt example.com.key > example.com.pem
# cp example.com.pem /etc/ssl/private/
```

5. **HAProxy를 구성합니다.**

```
# vim /etc/haproxy/haproxy.cfg
```

haproxy.cfg 의 global 및 defaults 섹션은 변경되지 않을 수 있습니다.defaults 섹션 후에는 다음 예와 같이 frontend 및 backend 섹션을 구성해야 합니다.

```
frontend http_web *:80
    mode http
    default_backend rgw

frontend rgw-https
  bind <insert vip ipv4>:443 ssl crt /etc/ssl/private/example.com.pem
  default_backend rgw

backend rgw
    balance roundrobin
    mode http
    server  rgw1 10.0.0.71:80 check
    server  rgw2 10.0.0.80:80 check
```

6. **enable/start haproxy**

```
# systemctl enable haproxy
# systemctl start haproxy
```

A.5. HAPROXY 설정 테스트

HAProxy 노드에서 keepalived 구성의 가상 IP 주소가 표시되는지 확인합니다.

```
$ ip addr show
```

ca plotari 노드에서 로드 밸런서 구성을 통해 게이트웨이 노드에 연결할 수 있는지 확인합니다. 예를 들면 다음과 같습니다.

```
$ wget haproxy
```

다음과 동일한 결과가 반환되어야 합니다.

```
$ wget rgw1
```

다음 내용이 포함된 index.html 파일을 반환하는 경우 구성이 제대로 작동합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
 <ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
   <ID>anonymous</ID>
   <DisplayName></DisplayName>
  </Owner>
  <Buckets>
  </Buckets>
 </ListAllMyBucketsResult>
```

부록 B. 버전 내역

고침 4.1-1 Wed Aug 7 2019 Steven Levine
7.7 GA 게시에 대한 문서 준비

고침 3.1-2 Thu Oct 4 2018 Steven Levine
7.6 GA 게시에 대한 문서 준비

고침 2.1-1 Thu Mar 15 2018 Steven Levine
7.5 GA 게시를 위한 문서 준비

고침 2.1-0 Thu Dec 14 2017 Steven Levine
7.5 베타 게시를 위한 문서 준비

고침 0.6-5 Wed Nov 22 2017 Steven Levine
7.4에 대한 업데이트 버전입니다.

고침 0.6-3 Thu Jul 27 2017 Steven Levine
7.4 GA 게시용 문서 버전입니다.

고침 0.6-1 Wed May 10 2017 Steven Levine
7.4 베타 게시에 대한 문서 준비.

고침 0.5-9 Mon Dec 5 2016 Steven Levine
7.3 버전입니다.

고침 0.5-7 Mon Oct 17 2016 Steven Levine
7.3 GA 게시 버전

고침 0.5-6 Thu Aug 18 2016 Steven Levine
7.3 베타 게시에 대한 문서 준비

고침 0.3-2 Mon Nov 9 2015 Steven Levine
7.2 GA 게시에 대한 문서 준비

고침 0.3-0 Wed Aug 19 2015 Steven Levine
7.2 베타 게시에 대한 문서 준비

고침 0.2-6 Mon Feb 16 2015 Steven Levine
7.1 GA 릴리스

고침 0.2-5 Thu Dec 11 2014 Steven Levine
7.1 베타 릴리스 버전

고침 0.2-4 Thu Dec 04 2014 Steven Levine
7.1 베타 릴리스 버전

고침 0.1-12 Tue Jun 03 2014 John Ha
7.0 GA 릴리스

고침 0.1-6 Mon Jun 13 2013 John Ha
Red Hat Enterprise Linux 7 베타 버전 빌드

고침 0.1-1 Wed Jan 16 2013 John Ha
이 문서의 Red Hat Enterprise Linux 6 버전에서 분기됨

색인

Symbols

A

F

H

K