Red Hat Enterprise Linux 7

ロードバランサーの管理

Keepalived および HAProxy の設定

Last Updated: 2021-01-18

Red Hat Enterprise Linux 7 ロードバランサーの管理

Keepalived および HAProxy の設定

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

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

法律上の通知

概要

ロードバランサーシステムを構築すると、Keepalived および HAProxy を用いて設定されたルーティングや負荷分散技術に対して特化した Linux Vertual Servers (LVS) を使用し、本番サービスに可用性および拡張性が高いソリューションが提供されます。本書では、Red Hat Enterprise Linux 7 のロードバランサー技術を使用した高パフォーマンスシステムおよびサービスの設定について説明します。

目次

第1章 ロードバランサーの概要

ロードバランサーは、複数の実サーバーの間で IP トラフィックを分散する統合されたソフトウェアコンポーネントです。Keepalived と HAProxy の 2 つの技術を使用してクラスターメンバーとクラスターサービスを監視します。Keepalived は *Linux 仮想サーバー* (LVS) を使用して負荷分散とフェイルオーバーのタスクをアティブルーターとパッシブルーターで実行します。一方、HAProxy は TCP および HTTP アプリケーションに対して負荷分散および高可用性サービスを実行します。

1.1. KEEPALIVED

keepalived デーモンは、アクティブおよびパッシブ LVS ルーターの両方で実行されます。**keepalived** で実行されるすべてのルーターは *Virtual Redundancy Routing Protocol* (VRRP) を使用します。アクティブルーターは定期的な間隔で VRRP アドバタイズメントを送信します。バックアップルーターが VRRP アドバタイズメントを正常に受信しなかった場合、新しいアクティブルーターが選出されます。

アクティブルーターでは、**keepalived** は実サーバーの負荷分散タスクも実行できます。

Keepalived は LVS ルーターに関連する制御プロセスです。起動時、デーモンは **/etc/keepalived/keepalived.conf** 設定ファイルを読み取る **systemctl** コマンドによって起動されます。アクティブルーターでは、**keepalived** デーモンは LVS サービスを開始し、設定されたトポロジーを基にサービスの健全性を監視します。アクティブルーターは VRRP を使用して周期的なアドバタイズメントをバックアップルーターへ送信します。バックアップルーターでは、VRRP インスタンスがアクティブルーターの稼働状況を判断します。ユーザーが設定した間隔の後にアクティブルーターのアドバタイズメントが失敗すると、Keepalived はフェイルオーバーを開始します。フェイルオーバー中に仮想サーバーがクリアになります。新しいアクティブルーターは *仮想 IP アドレス* (VIP) を制御して、ARP メッセージを送信し、IPVS テーブルエントリー (仮想サーバー) を設定します。さらに、ヘルスチェックを開始し、VRRP アドバタイズメントの送信を開始します。

Keepalived は、TCP が接続ベースのデータ送信を実行するレイヤー4 (トランスポート層) でフェイルオーバーを実行します。実サーバーが単純なタイムアウト TCP 接続に応答しなかった場合、**keepalived** はサーバーの失敗を検出し、そのサーバーをサーバープールから削除します。

1.2. HAPROXY

HAProxy は、インターネットに接続されたサービスや Web ベースのアプリケーションなどの HTTP および TCP ベースのサービスに負荷分散されたサービスを提供します。選択したロードバランサースケジューリングアルゴリズムに応じて、**haproxy** は 1 つの仮想サーバーとして動作する複数の実サーバーのプール全体で、複数のイベントを数千個の接続で処理できます。スケジューラーは接続の量を判断し、非加重のスケジュールで均等に割り当てするか、加重を認識するアルゴリズムで高容量の処理が可能なサーバーにより多くの接続量を割り当てします。

HAProxy では、ユーザーは複数のプロキシーサービスを定義でき、プロキシーに対してトラフィックの負荷分散サービスを実行します。プロキシーは、1 つのフロントエンドシステムと 1 つ以上のバックエンドシステムで構成されます。フロントエンドシステムは IP アドレス (VIP)、プロキシーがリッスンするポート、および特定のプロキシーに使用するバックエンドシステムを定義します。

バックエンドシステムは実サーバーのプールで、負荷分散アルゴリズムを定義します。

HAProxy はレイヤー 7 (アプリケーション層) で負荷分散管理を実行します。ほとんどの場合で、管理者は HTTP ベースの負荷分散 (ビジネスの継続に高可用性インフラストラクチャーが必要となる本番 Web アプリケーションなど) に HAProxy をデプロイします。

1.3. KEEPALIVED および HAPROXY

管理者は Keepalived と HAProxy を両方使用して堅牢でスケーラブルな高可用性環境を実現できます。Keepalived のフェイルオーバーサービスとともに、HAProxy のスピードとスケーラビリティーを使用して HTTP およびその他の TCP ベースのサービスに負荷分散を実行すると、実サーバー間で負荷を分散して可用性を高めることができ、バックアップルーターへのフェイルオーバーによってルーターが利用できなくなってもサービスが継続されます。

第2章 KEEPALIVED の概要

Keepalived は *アクティブLVS ルーター* や1つ以上の任意の *バックアップLVS ルーター* で実行されます。アクティブ LVS ルーターには 2 つの役割があります。

- 複数の実サーバー全体の負荷分散
- それぞれの実サーバー上にあるサービスの整合性チェック

アクティブ (マスター) ルーターは、*Virtual Router Redundancy Protocol* (VRRP) を使用してアクティブな状態をバックアップルーターに通知します。マスタールーターは、通常の間隔でアドバタイズメントを送信する必要があります。アクティブルーターがアドバタイズメントの送信を停止すると、新しいマスターが選出されます。

注記

Red Hat は、使用する VRRP バージョンをその設定が変更する keepalived のローリング更新をサポートしていません。すべてのルーターは、keepalived ロードバランサー設定で同じバージョンの VRRP を実行している必要があります。VRRP バージョンの不一致により、次のメッセージが表示されます。

```
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 Hat は、互換性の問題を回避するために、すべてのシステムで同じ keepalived バージョンを実行し、可能であれば keepalived 設定を同一にする必要があります。

2.1. 基本的な KEEPALIVED ロードバランサーの設定

図2.1「基本的なロードバランサー設定」 は、2 層で構成されるシンプルな Keepalived ロードバランサー設定を示しています。第1層には、1つのアクティブ LVS ルーターと、複数のバックアップ LVS ルーターがあります。各 LVS ルーターには、インターネット上とプライベートネットワーク上にそれぞれ1つのネットワークインターフェースがあり、これらのネットワーク間のトラフィックを調整できるようになっています。この図の例では、アクティブルーターは *ネットワークアドレス変換* (*Netowrk Address Translation*、*NAT*) を使用してインターネットから第 2 層にある実サーバーへトラフィックを移動し、必要なサービスを提供します。そのため、この例の実サーバーは専用のプライベートネットワークセグメントに接続され、アクティブ LVS ルーターを介してすべてのパブリックトラフィックを送受信します。外部では、サーバーは1つのエンティティーのように見えます。

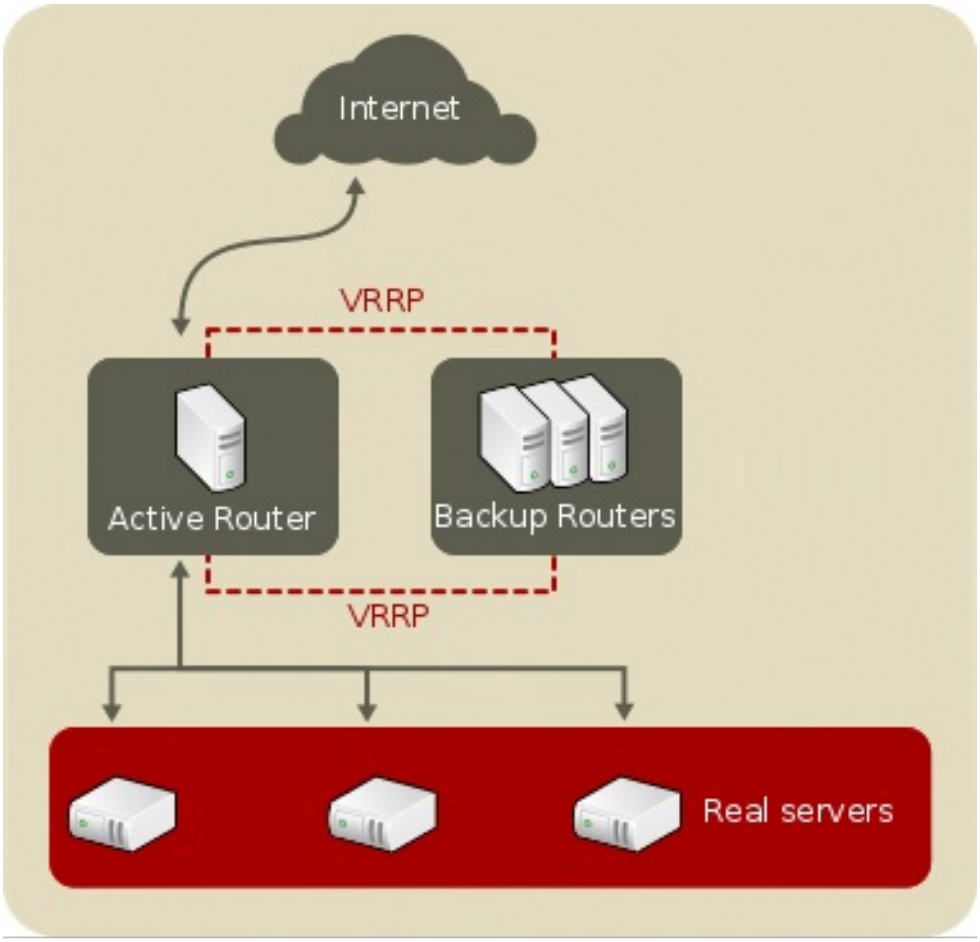


図2.1 基本的なロードバランサー設定

LVS ルーターへ送信されたサービス要求は*仮想 IP* アドレス (*VIP*) 宛になります。このアドレスはパブリックにルーティング可能なアドレスで、サイトの管理者が www.example.com などの完全修飾ドメイン名を関連付けし、1 つ以上の*仮想サーバー*へ割り当てます。仮想サーバーは、特定の仮想 IP でリッスンするよう設定されたサービスです。VIP アドレスはフェイルオーバー中に、ある LVS ルーターから別の LVS ルーターに移行するため、その IP アドレスで存在を維持します。VIP は**フローティング IP アドレス**とも呼ばれます。

LVS ルーターをインターネットに接続するデバイスに VIP アドレスが割り当てられることがあります。たとえば、**eth0** がインターネットに接続されている場合、複数の仮想サーバーを **eth0** に割り当てることができます。この代わりに、各仮想サーバーをサービスごとに個別のデバイスへ関連付けることができます。たとえば、HTTP トラフィックを 192.168.1.111 の **eth0** で処理し、FTP トラフィックを 192.168.1.222 の **eth0** で処理することができます。

1 つのアクティブなルーターと 1 つのパッシブなルーターが関係するデプロイメントの場合、アクティブサーバーの役目は仮想 IP アドレスから実サーバーへサービス要求をリダイレクトすることです。リダイレクトは、「keepalived スケジューリングの概要」 に記載されている 8 つのサポートされる負荷分散アルゴリズムの 1 つを基にして実行されます。

アクティブルーターは、シンプルな TCP 接続、HTTP、および HTTPS の 3 つのビルトインヘルスチェックを使用して実サーバーで特定サービスの全体的な健全性を動的に監視します。TCP 接続では、アクティブルーターは特定のポートで実サーバーに接続できることを周期的にチェックします。HTTP および HTTPS では、アクティブルーターは周期的に実サーバーの URL をフェッチし、コンテンツを検証します。

バックアップルーターはスタンバイシステムの役割を担います。ルーターのフェイルオーバーは VRRP によって処理されます。起動時、すべてのルーターはマルチキャストグループに参加します。このマルチキャストグループは、VRRP アドバタイズメントの送受信に使用されます。VRRP は優先度ベースの

プロトコルであるため、優先度が最も高いルーターがマスターとして選出されます。ルーターがマスターとして選出されると、定期的な間隔で VRRP アドバタイズメントをマルチキャストグループへ送信します。

一定の期間内 (アドバタイズメントの間隔を基にした) にバックアップルーターがアドバタイズメントを受信しなかった場合、新しいマスターが選出されます。新しいマスターは VIP を引き継ぎ、*アドレス解決プロトコル* (Address Resolution Protocol、ARP) メッセージを送信します。ルーターがアクティブなサービスを返すと、バックアップまたはマスターになります。この動作はルーターの優先度によって決まります。

図2.1「基本的なロードバランサー設定」 で使われているシンプルな二層設定は、静的な Web ページのように頻繁に変更されないデータに適しています。これは、個別の実サーバーは各ノード間で自動的にデータを同期しないためです。

2.2. 3 層の KEEPALIVED ロードバランサーの設定

図2.2「3 層のロードバランサーの設定」 は、典型的な 3 層の Keepalived ロードバランサートポロジーを示しています。この例では、アクティブ LVS ルーターは要求をインターネットから実サーバーのプールにルーティングします。その後、実サーバーはネットワーク上で共有データソースにアクセスします。

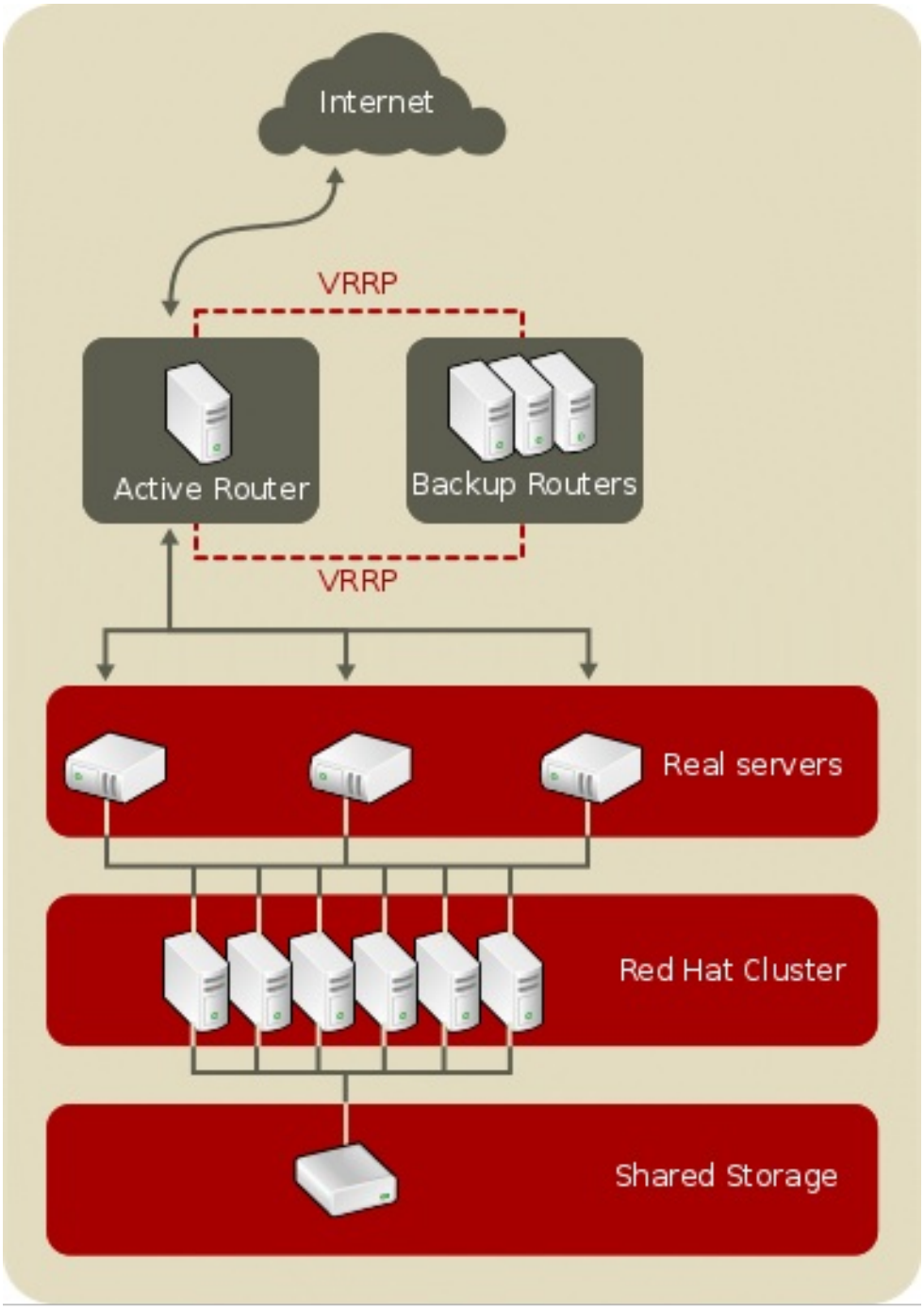


図2.2 3 層のロードバランサーの設定

この設定は、アクセス可能なデータが中央の高可用性サーバーに保存され、エクスポートされた NFS ディレクトリーまたは Samba による共有を用いて各実サーバーによってアクセスされるビジーな FTP サーバーの場合に適しています。このトポロジーは、トランザクションのために中央の高可用性データベースにアクセスする Web サイトにも推奨されます。また、ロードバランサーでアクティブ-アクティブ設定を使用すると、管理者はこれら両方の役割を同時に果たす高可用性クラスターを設定することもできます。

上記の例では、第 3 層でロードバランサーを使う必要はありませんが、高可用性のソリューションを使わないと重大な単一障害点となります。

2.3. KEEPALIVED スケジューリングの概要

Keepalived を使用すると、さまざまなスケジューリングアルゴリズムがサポートされるため、実サーバー間でトラフィックを分散する柔軟性が高くなります。階層的な DNS やクライアントマシンの

キャッシングによって負荷が不均等になる *ラウンドロビン DNS* などの柔軟性が低い方法よりも、負荷分散に優れています。さらに、ネットワークパケットレベルで負荷分散を行うと計算のオーバーヘッドが最小限になり、スケーラビリティーが向上するため、LVS ルーターが使用する低レベルのフィルタリングは、アプリケーションレベルの要求転送よりも優れています。

割り当てられた加重値を使用すると、各マシンに任意の優先度を提供します。このようなスケジューリングを使用すると、さまざまなハードウェアとソフトウェアの組み合わせを使用して実サーバーのグループを作成でき、アクティブなルーターは負荷を実サーバーへ均等に分散できます。

Keepalived のスケジューリングメカニズムは、*IP 仮想サーバー* または *IPVS* モジュールと呼ばれるカーネルパッチの集合によって提供されます。このモジュールは *レイヤー 4* (*L4*) トランスポート層スイッチングを有効にします。これは、単一 IP アドレス上で複数のサーバーとうまく機能するように設計されています。

パケットの追跡や実サーバーへのルーティングを効率的に行うため、IPVS はカーネルに *IPVS テーブル*を構築します。このテーブルは、アクティブ LVS ルーターによって使用され、要求を仮想サーバーアドレスからプールの実サーバーへリダイレクトしたり、プールの実サーバーから戻したりします。

2.3.1. keepalived スケジューリングアルゴリズム

IPVS テーブルの構造は、管理者が仮想サーバーに対して選択するスケジューリングアルゴリズムによって異なります。クラスター化できるサービスのタイプや、これらのサービスがスケジュールされる方法で柔軟性を最大限にするため、Keepalived は以下のスケジューリングアルゴリズムをサポートします

ラウンドロビンスケジューリング (Round-Robin Scheduling)

要求を順番に実サーバーのプールで分配します。このアルゴリズムを使用すると、容量や負荷に関係なく実サーバーはすべて同等に扱われます。このスケジューリングモデルはラウンドロビン DNS と似ていますが、ホストベースではなくネットワークベースであるため、粒度がより細かくなります。また、ロードバランサーのラウンドロビンスケジューリングでは、キャッシュされた DNS クエリーが原因で負荷が不均等になることはありません。

加重ラウンドロビンスケジューリング (Weighted Round-Robin Scheduling)

各要求を順番に実サーバーのプールで分散しますが、容量の多いサーバーにより多くのジョブを割り当てます。容量は、ユーザーが割り当てる加重係数によって示され、動的な負荷情報によって調整されます。

プール内の実サーバー間で処理能力に大幅な違いがある場合は、重み付きラウンドロビンスケジューリングが適しています。ただし、要求負荷が大きく変化する場合は、重みの大きいサーバーが割り当て以上の要求に応じる可能性があります。

最小接続 (Least-Connection)

実際の接続が少ない実サーバーにより多くの要求を振り分けます。IPVS テーブルで実サーバーへのライブ接続を継続的に追跡するため、Least-Connection は動的なスケジューリングアルゴリズムになります。要求負荷の変化が大きい場合に適しています。このアルゴリズムは各メンバーノードの処理能力がほぼ同じで大差がないような実サーバープールに最適です。サーバーグループの処理能力が異なる場合は weighted least-connection スケジューリングの方が適しています。

加重最小接続 (Weighted Least-Connections)

容量と比較してより少ないアクティブな接続を持つサーバーにより多くの要求を分散します。容量は、ユーザーが割り当てる加重係数によって示され、動的な負荷情報によって調整されます。実際のサーバープールに異なる容量のハードウェアが含まれる場合、加重が追加されるこのアルゴリズムが適しています。

ローカリティーベースの最小接続スケジューリング (Locality-Based Least-Connection Scheduling)

接続先 IP に対して相対的にアクティブな接続が少ないサーバーにより多くの要求を振り分けます。このアルゴリズムは、プロキシキャッシュサーバーのクラスターで使用するために設計されています。サーバーがキャパシティーを超えておらず、負荷が半分の別のサーバーがない場合、IP アドレスのパケットをそのアドレスのサーバーに送信します。サーバーがキャパシティーを超え、負荷が半分のサーバーがある場合は、IP アドレスを負荷が最も少ない実サーバーに割り当てます。

複製スケジューリングを伴うローカリティベースの最小接続スケジューリング (Locality-Based Least-Connection Scheduling with Replication Scheduling)

接続先 IP に対して相対的にアクティブ接続が少ないサーバーにより多くの要求を振り分けます。このアルゴリズムは、プロキシキャッシュサーバーのクラスターで使用するために設計されています。ローカリティベースの最小接続スケジューリングとの違いは、ターゲットの IP アドレスが実サーバーノードのサブセットにマッピングされる点です。その後に要求は、このサブセット内で接続が最も少ないサーバーに送信されます。接続先 IP のノードがすべてキャパシティーを上回っている場合、実サーバーのプール全体で接続が一番少ないサーバーをその接続先 IP の実サーバーのサブセットに追加することで、接続先 IP アドレス向けに新たなサーバーを複製します。すると、負荷の最も高いノードが実サーバーのサブセットから外され、過剰な複製が抑制されます。

接続先ハッシュスケジューリング (Destination Hash Scheduling)

静的ハッシュテーブル内の接続先 IP を検索して、実サーバーのプールに要求を振り分けます。このアルゴリズムは、プロキシキャッシュサーバーのクラスターで使用するために設計されています。

ソースハッシュスケジューリング (Source Hash Scheduling)

静的ハッシュテーブル内のソース IP を検索して、実サーバーのプールに要求を振り分けます。このアルゴリズムは、複数のファイアウォールがある LVS ルーター向けに設計されています。

最短予測遅延 (Shortest Expected Delay)

サーバー上の接続数を割り当てられた加重値で割った値を基して、最も短い遅延が期待されるサーバーへ接続要求を分配します。

キューなし (Never Queue)

最初に接続要求を見つけ、アイドル状態または接続がないサーバーへ接続要求を送信する 2 面のスケジューラーです。アイドル状態のサーバーがない場合、スケジューラーはデフォルトで **最短予測遅延 (Shortest Expected Delay)** と同様に最も遅延が少ないサーバーを選択します。

2.3.2. サーバーの加重値とスケジューリング

ロードバランサーの管理者は、実際のサーバープールの各ノードに*加重値*を割り当てできます。この加重値は整数値で、**加重値を認識する**スケジューリングアルゴリズム (加重最小接続など) で考慮されます。また、LVS ルーターが異なる容量を持つハードウェアで負荷を均等にできるようにします。

加重値は、それぞれに対する相対的な割合として機能します。たとえば、ある実サーバーの加重値が 1 で別のサーバーの加重値が 5 の場合、前者が 1 回接続されるごとに後者が 5 回接続されます。実サーバーの加重値のデフォルト値は 1 です。

実サーバープール内でそれぞれ異なるハードウェア構成のノードに重みを付けるとクラスターでの負荷分散をより効率的に行う場合に役立ちますが、 weighted least-connection スケジューリングで仮想サーバーが設定されている際、任意の実サーバーがその実サーバープールに挿入されると、一時的に不均衡が生じる場合があります。例えば、実サーバープールに 3 台のサーバーがあったとします。サーバー A と B に 1 の重みが付けられ、サーバー C には 2 の重みが付けられていたとします。サーバー C が何らかの理由でダウンした場合、放棄された負荷がサーバー A と B に均等に分散されます。しか

し、サーバー C がオンラインに復帰すると、LVS ルーター側でサーバー C の接続がまったくないと判断され、サーバー A および B と同等になるまで着信要求をすべてサーバー C に集中的に振り分けることになります。

2.4. ルーティングメソッド

Red Hat Enterprise Linux は*ネットワークアドレス変換* (*NAT ルーティング*) または Keepalived のダイレクトルーティングを使用します。これにより、利用できるハードウェアを活用し、ロードバランサーを既存のネットワークに統合する場合に柔軟性が大変高くなります。

2.4.1. NAT ルーティング

図2.3「NAT ルーティングを実装したロードバランサー」 は、ロードバランサーが NAT ルーティングを使用して、インターネットとプライベートネットワークの間で要求を移動する様子を示しています。

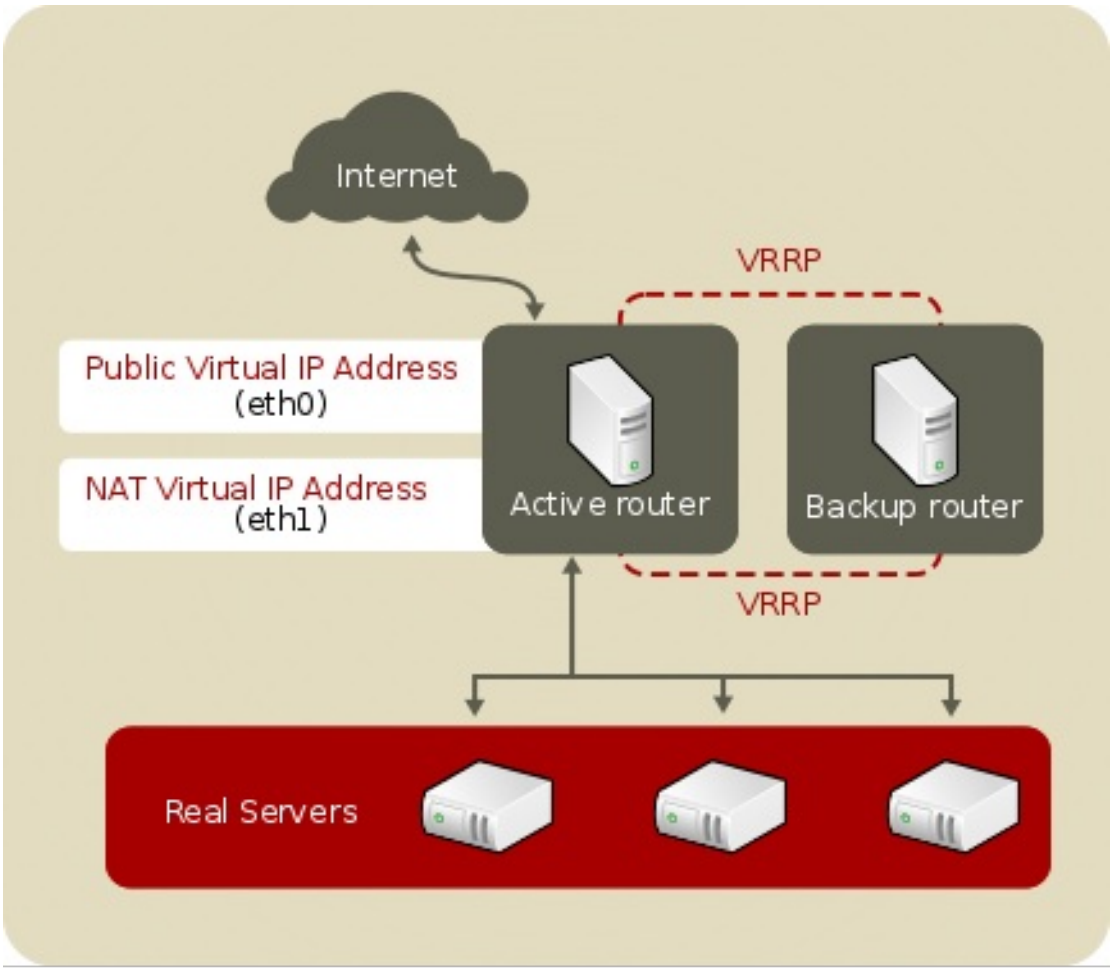


図2.3 NAT ルーティングを実装したロードバランサー

この例では、アクティブ LVS ルーターに NIC が 2 つあります。インターネットの NIC では *リアル IP アドレス*とフローティング IP アドレスが eth0 にあります。プライベートネットワークインターフェースの NIC はリアル IP アドレスとフローティング IP アドレスが eth1 にあります。フェイルオーバーが発生すると、インターネットに向けられている仮想インターフェースと仮想インターフェースに向けられているプライベートネットワークが同時にバックアップ LVS ルーターへ引き継ぎされます。プライベートネットワークにあるすべての実サーバーは、NAT ルーターのフローティング IP をアクティブ LVS ルーターと通信するデフォルトのルートとして使用します。こうすることで、インターフェースからの要求に応答する能力が維持されます。

この例では、LVS ルーターのパブリックフローティング IP アドレスとプライベート NAT フローティン

グ IP アドレスが物理 NIC に割り当てられます。フローティング IP アドレスを LVS ルーターノード上のそれぞれの物理デバイスに関連付けることは可能ですが、3 つ以上の NIC を使用する必要はありません。

このトポロジーを使うと、アクティブ LVS ルーターは要求を受信して適切なサーバーにルーティングします。実サーバーはその要求を処理してパケットを LVS ルーターに返します。LVS ルーターはネットワークアドレス変換を使ってパケット内の実サーバーのアドレスを LVS ルーターのパブリック VIP アドレスに置き換えます。実サーバーの本当の IP アドレスは要求を行っているクライアントからは見えないよう隠しているため、*IP マスカレード* と呼ばれます。

NAT ルーティングを使用する場合は、実サーバーにするマシンの種類や稼働させるオペレーティングシステムの種類に制限はありません。ただし、発信要求および着信要求のいずれも LVS ルーターで処理しなければならないため、大規模なクラスター導入の場合には LVS ルーターがボトルネックとなる場合があります。

\ipvs モジュールは、iptables および ip6tables NAT とは独立した独自の NAT ルーティングを使用します。これにより、実サーバーが **/etc/keepalived/keepalived.conf** ファイルで DR ではなく NAT に対して設定されている場合に、IPv4 と IPv6 NAT の両方に対応します。

2.4.2. ダイレクトルーティング

ダイレクトルーティングを使用するロードバランサー設定を構築すると、他のロードバランサーネットワーキングトポロジーよりもパフォーマンス上のメリットが大きくなります。ダイレクトルーティングでは、実サーバーは要求元のユーザーに対して直接パケットを処理およびルーティングでき、すべての送信パケットを LVS ルーター経由で渡しません。ダイレクトルーティングでは、LVS ルーターが受信パケットのみを処理するよう制限し、ネットワークパフォーマンスの問題が発生する可能性を低減します。

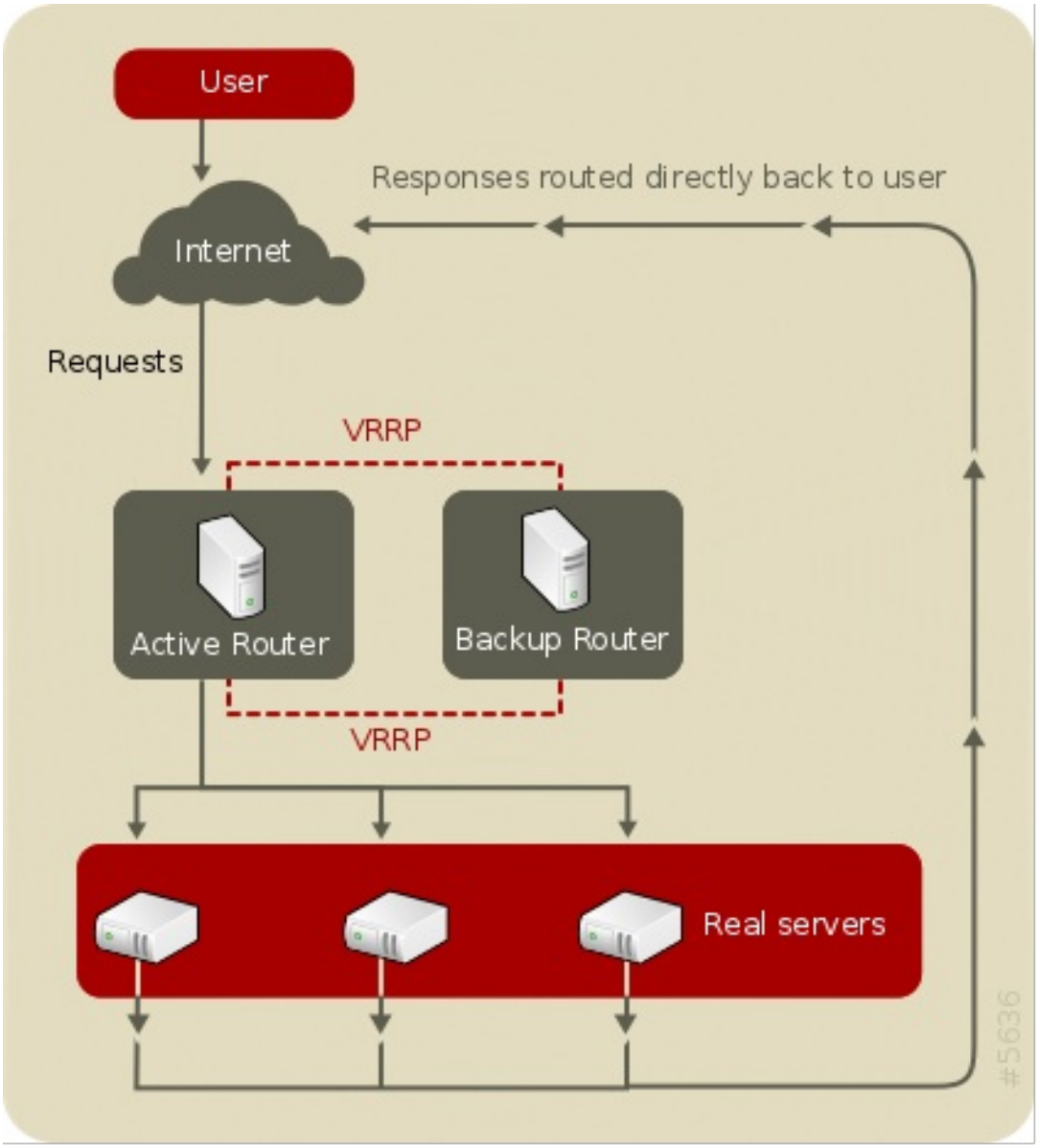


図2.4 ダイレクトルーティングで実装されたロードバランサー

典型的なダイレクトルーティングのロードバランサー設定では、LVS ルーターは仮想 IP (VIP) 経由で受信サーバー要求を受け取り、スケジューリングアルゴリズムを使用して要求を実サーバーへルーティングします。実サーバーは要求を処理し、LVS ルーターを通さずに応答を直接クライアントに送信します。ネットワークの負荷が高い状態で LVS ルーターが送信パッケージを実サーバーからクライアントへルーティングするとボトルネックになりますが、 このルーティング方法では LVS ルーターがこのような処理をしなくても実サーバーを追加できるため、スケーラビリティーが向上します。

2.4.2.1. ダイレクトルーティングと ARP 制限

ロードバランサーでダイレクトルーティングを使用する利点は多くありますが、制限もあります。最も一般的な問題は、*アドレス解決プロトコル* (ARP) に関する問題です。

インターフェース上のクライアントは要求を IP アドレスに送信します。ネットワークルーターは ARP を使って IP アドレスをマシンの MAC アドレスに関連付けることで要求を宛先に送信します。ARP 要求がネットワークに接続されているすべてのマシンにブロードキャストされ、IP アドレスと MAC アドレスの正しい組み合わせを持つマシンがパケットを受け取ることになります。IP と MAC の関連性は ARP キャッシュに保存され、定期的に消去と再保存が行われます (通常 15 分ごと)。

ダイレクトルーティングのロードバランサー設定における ARP 要求の問題は、IP アドレスへのクライアント要求が処理されるためにはその要求が MAC アドレスに関連付けされている必要があるため、

ロードバランサーシステムの仮想 IP アドレスも MAC に関連付けされている必要があることです。しかし、LVS ルーターと実サーバーは同じ VIP を持っているため、ARP 要求はその VIP に関連付けられているすべてのマシンへブロードキャストされます。これにより、VIP が直接実サーバーの 1 つに関連付けられて直接要求を処理する問題や、完全に LVS ルーターを迂回してロードバランサー設定の目的に反する問題など、複数の問題が発生することがあります。

この問題を解決するには、着信要求が実サーバーの 1 つではなく、常に LVS ルーターに送信されるようにしてください。これには、ARP 要求または IP パケットをフィルターします。ARP のフィルターは **arptables** ユーティリティーを使用して実行することができ、IP パケットは **iptables** または **firewalld** を使用してフィルターできます。この 2 つの方法の違いは以下のとおりです。

- ARP のフィルター方法では、実サーバーに到達する要求をブロックします。これにより、ARP が実サーバーで VIP と関連することを防ぎ、アクティブな仮想サーバーが MAC アドレスで応答するようにします。

- IP パケットのフィルター方法では、他の IP アドレスでパケットを実サーバーにルーティングできます。最初から実サーバーで VIP を設定しないと、ARP の問題を完全に回避できます。

2.5. KEEPALIVED の永続性およびファイアウォールマーク

状況によっては、負荷分散アルゴリズムを使用して利用可能な最良のサーバーへ要求を送信せずに、クライアントが同じ実サーバーに繰り返し再接続する方が望ましいことがあります。このような場合の例には、マルチスクリーン Web フォーム、クッキー、SSL、および FTP 接続などがあります。このようなケースでは、同じサーバーがトランザクションを処理し、コンテキストを保持しないとクライアントが適切に動作しないことがあります。Keepalived は、*永続性*と *ファイアウォールマーク*の 2 つの機能でこれに対応します。

2.5.1. 永続性

永続性が有効になっていると、タイマーのように動作します。クライアントがサービスに接続すると、ロードバランサーは指定期間中は最後の接続を記憶します。同じクライアント IP アドレスが同じ期間内に再接続すると、以前接続したサーバーへ送信され、負荷分散メカニズムが無視されます。接続が指定期間外で発生すると、対応するスケジューリングルールに従って処理されます。

また、永続性を使うと、管理側でサブネットマスクを指定して、どのアドレスがより高い永続性を持つかを管理するツールとして、クライアント IP アドレステストに適用することができます。こうすることで、接続をそのサブネットにグループ化できます。

通信に複数のポートを使用する FTP などのプロトコルの場合、宛先が別々のポートの接続をグループ化することがとても重要な場合があります。ただし、宛先が別々のポートの接続をグループ化する上で発生する問題に対処する場合、永続性は最も効率的な方法とは言えません。このような場合には *ファイアウォールマーク* を使用するのが最適です。

2.5.2. ファイアウォールマーク

ファイアウォールマークは、プロトコルまたは関連するプロトコルのグループに使用されるポートを簡単かつ効率的にグループ化する方法です。たとえば、EC (E コマース) サイトの稼働が目的でロードバランサーがデプロイされた場合、ファイアウォールマークを使用してポート 80 の HTTP 接続と、ポート 443 のセキュアな HTTPS 接続をバンドル化できます。各プロトコルの仮想サーバーに同じファイアウォールマークを割り当てると、接続が開かれた後に LVS ルーターがすべての要求を同じ実サーバーへ転送するため、トランザクション状態の情報を保持できます。

ファイアマークは効率的で簡単に使用できるため、ロードバランサーの管理者は可能な限り永続性の代わりにファイアウォールマークを使用して接続をグループ化するようにしてください。しかし、クライアントが十分な期間同じサーバーに再接続されるようにするため、ファイアウォールマークとともに永

続性を仮想サーバーに追加するようにしてください。

第3章 KEEPALIVED のロードバランサー要件の設定

keepalived を使用するロードバランサーは、LVS ルーターと実サーバーの 2 つの基本グループで構成されます。単一障害点を回避するため、各グループに 2 つ以上のメンバーが含まれるようにしてください。

LVS ルーターグループは、Red Hat Enterprise Linux を実行している同一または非常に似ている 2 つのシステムで構成する必要があります。そのうちの 1 つはアクティブ LVS ルーターとして機能し、もう 1 つはホットスタンバイモードで待機するので、この 2 つができるだけ同じ機能を備えている必要があります。

実際のサーバーグループのハードウェアを選択および設定する前に、3 つのロードバランサートポロジーのどれを使用するかを決定します。

3.1. NAT ロードバランサーネットワーク

NAT トポロジーを使用すると、既存のハードウェアを柔軟に活用できますが、プールから送受信されるすべてのパケットはロードバランサールーターを通過するため、大量の負荷を処理する機能が制限されます。

ネットワークレイアウト

NAT ルーティングを使用するロードバランサーのトポロジーは、パブリックネットワークへのアクセスポイントが 1 つのみ必要なため、ネットワークレイアウトの観点から見ると最も設定が簡単です。実サーバーはプライベートネットワーク上にあり、LVS ルーター経由ですべての要求を返します。

ハードウェア

NAT トポロジーでは、実サーバーは LVS ルーターのみに応答するため、実サーバーごとに 1 つの NIC のみが必要になります。一方で LVS ルーターは 2 つのネットワークの間でトラフィックを送信するため、NIC が 2 つずつ必要になります。このトポロジーでは LVS ルーターの部分がネットワークのボトルネックとなるため、ギガビットイーサネット NIC を LVS ルーターで使うことで、LVS ルーターが処理可能な帯域幅を拡げることができます。ギガビットイーサネットを LVS ルーターで使用する場合は、実サーバーを LVS ルーターに接続しているスイッチは、負荷を効率的に処理するために少なくとも 2 つのギガビットイーサネットポートが必要になります。

ソフトウェア

NAT トポロジーでは、設定によっては **iptables** を使用する必要があるため、Keepalived 外部で多くのソフトウェアを設定する場合があります。特に、FTP サービスやファイアウォールマークを使用する場合は、要求を適切にルーティングするために LVS ルーターを追加で手動設定する必要があります。

3.1.1. NAT を使用するロードバランサーのネットワークインターフェース設定

NAT を使用するロードバランサーを設定するには、最初に LVS ルーターのパブリックネットワークとプライベートネットワークのネットワークインターフェースを設定する必要があります。この例では、LVS ルーターのパブリックインターフェース (**eth0**) を 203.0.113.0/24 ネットワークに設定し、実サーバーにリンクするプライベートインターフェース (**eth1**) を 10.11.12.0/24 ネットワークに設定します。

重要

本書の執筆時点では **NetworkManager** サービスはロードバランサーに対応していません。IPv6 アドレスが SLAAC に割り当てられた場合、IPv6 VIP は動作しないことが判明しています。そのため、この例では設定ファイルと **network** サービスを使用します。

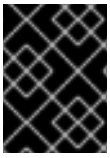
アクティブもしくは *プライマリー* LVS ルーターノードでは、パブリックインターフェースのネットワーク設定ファイルである **/etc/sysconfig/network-scripts/ifcfg-eth0** は以下のようになります。

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=203.0.113.9
NETMASK=255.255.255.0
GATEWAY=203.0.113.254
```

LVS ルーター上のプライベート NAT インターフェースの **/etc/sysconfig/network-scripts/ifcfg-eth1** 設定ファイルは、以下のようになります。

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

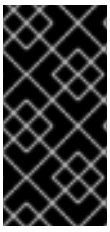
この例では、LVS ルーターのパブリックインターフェースの VIP は 203.0.113.10 となり、NAT もしくはプライベートインターフェースの VIP は 10.11.12.10 となります。実サーバーが要求を NAT インターフェースの VIP に返送することが重要になります。



重要

本項のイーサネットインターフェース設定の例は、LVS ルーターのリアル IP アドレス向けの例で、フローティング IP アドレス向けの例ではありません。

プライマリー LVS ルーターノードのネットワークインターフェースを設定した後、バックアップ LVS ルーターの実ネットワークインターフェースを設定します。IP アドレスがネットワーク上の他の IP アドレスと競合しないように注意してください。



重要

バックアップノード上の各インターフェースが、プライマリーノード上のインターフェースと同じネットワークに対応するようにしてください。たとえば、eth0 がプライマリーノード上でパブリックネットワークに接続する場合、バックアップノードでもパブリックネットワークに接続するようにする必要があります。

3.1.2. 実サーバー上でのルーティング

NAT トポロジーで実サーバーネットワークインターフェースを設定する場合に最も重要なのは、LVS ルーターの NAT フローティング IP アドレス用にゲートウェイを設定することです。この例では、ゲートウェイのアドレスは 10.11.12.10 となります。

注記

実サーバー上でのネットワークインターフェース設定が完了すると、マシンは他の方法でパブリックネットワークに ping したり接続したりすることができなくなります。これは正常なことです。しかし、LVS ルーターのプライベートインターフェースの実 IP (この場合は 10.11.12.9) に ping することはできます。

実サーバーの **/etc/sysconfig/network-scripts/ifcfg-eth0** 設定ファイルは以下のようになります。

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
IPADDR=10.11.12.1
NETMASK=255.255.255.0
GATEWAY=10.11.12.10
```

警告

GATEWAY= 行で実サーバーに複数のネットワークインターフェースが設定されている場合、最初に起動したネットワークインターフェースがゲートウェイを取得します。そのため、**eth0** と **eth1** の両方が設定され、**eth1** がロードバランサーに使用されている場合、実サーバーは要求を適当にルーティングしないことがあります。

/etc/sysconfig/network-scripts/ ディレクトリー内のネットワーク設定ファイルで **ONBOOT=no** と設定するか、最初に表示されるインターフェースでゲートウェイを適切に設定して、外部のネットワークインターフェースをオフにすることが推奨されます。

3.1.3. LVS ルーターでの NAT ルーティングの有効化

クラスター化されたサービスが 1 つのポート (例: ポート 80 の HTTP) のみを使用する簡単な NAT ロードバランサー設定の場合、LVS ルーター上でパケット転送のみを有効にすれば実サーバーとその外部との間で適切に要求をルーティングできます。しかし、ユーザーセッション中に同じ実サーバーに移動するため、クラスター化されたサービスが複数のポートを必要とする場合は、他の設定が必要になります。

LVS ルーターで転送が有効になり、実サーバーが設定され、クラスター化されたサービスが稼働したら、**keepalived** を使用して IP の情報を設定します。

警告

手作業でネットワーク設定ファイルを編集したり、ネットワーク設定ツールを使用したりして **eth0** または **eth1** のフローティング IP を設定しないでください。必ず **keepalived.conf** ファイルで設定してください。

設定が終了したら、 **keepalived** サービスを開始します。サービスの稼働後、アクティブ LVS ルーターは要求を実サーバーのプールへルーティングします。

3.2. ダイレクトルーティングを使用するロードバランサー

ダイレクトルーティングを使用すると、実サーバーは LVS ルーターを介して送信パケットを渡さずに、直接パケットを処理して要求元ユーザーにルーティングできます。ダイレクトルーティングでは、実サーバーが LVS ルーターがあるネットワークセグメントに物理的に接続している必要があり、送信パケットを処理およびルーティングできる必要があります。

ネットワークレイアウト

ダイレクトルーティングのロードバランサー設定では、LVS ルーターが受信要求を受け取り、要求を処理する適切な実サーバーへルーティングする必要があります。その後、実サーバーは応答を**直接**クライアントへルーティングする必要があります。たとえば、クライアントがインターネット上にあり、LVS ルーター経由でパケットを実サーバーへ送信する場合、実サーバーはインターネット経由で直接クライアントに接続できる必要があります。これを実現するには、実サーバーのゲートウェイを設定し、パケットをインターネットに渡します。サーバープールの各サーバーは独自のゲートウェイを持ちます (各ゲートウェイはインターネットへの独自の接続を持ちます)。そのため、スループットやスケーラビリティーを最大限にします。しかし、一般的なロードバランサーの設定では実サーバーは 1 つのゲートウェイ (よって 1 つのネットワーク接続) を経由して通信できます。

ハードウェア

ダイレクトルーティングを使用するロードバランサーシステムのハードウェア要件は、他のロードバランサートポロジーと似ています。LVS ルーターは Red Hat Enterprise Linux を実行して受信要求を処理し、実サーバーの負荷分散を実行する必要がありますが、実サーバーが適切に機能するには Linux マシンである必要はありません。各 LVS ルーターには 1 つまたは 2 つの NIC が必要です (バックアップルーターがあるかによって異なります)。NIC を 2 つ使用すると設定が容易になり、受信要求は 1 つの NIC によって処理され、別の NIC によってパケットが実サーバーへルーティングされるため、トラフィックを区別できます。

実サーバーは LVS ルーターを迂回して送信パケットを直接クライアントに送信するため、インターネットへのゲートウェイが必要となります。パフォーマンスと可用性を最大化するには、クライアントが接続しているネットワーク (インターネットやイントラネットなど) に専用接続がある独自のゲートウェイに実サーバーを接続します。

ソフトウェア

ダイレクトルーティングでロードバランサーを使用するときに ARP の問題が発生する場合は、特に **keepalived** 以外の設定を行う必要があります。詳細は 「arptables を使用したダイレクトルーティング」 または 「iptables を使用したダイレクトルーティング」 を参照してください。

3.2.1. arptables を使用したダイレクトルーティング

arptables を使用してダイレクトルーティングを設定するには、実サーバーで仮想 IP アドレスが設定され、直接パケットをルーティングできる必要があります。VIP の ARP 要求は実サーバーによって完全に無視されます。VIP を含む送信される ARP パケットは、VIP ではなく実サーバーの IP が含まれるよう分割されます。

arptables メソッドを使用すると、アプリケーションを実サーバーが接続する個別の VIP またはポートにバインドすることができます。たとえば、**arptables** メソッドを使用すると Apache HTTP Server の複数のインスタンスを実行し、システム上の異なる VIP に明示的にバインドすることができます。

しかし、**arptables** メソッドを使うと、標準の Red Hat Enterprise Linux システム設定ツールを使用して起動時に VIP を開始する設定ができません。

それぞれの仮想 IP アドレスの ARP 要求を無視するように実サーバーを設定するには、以下の手順を実行します。

1. 実サーバー上で仮想 IP アドレスの ARP テーブルのエントリーを作成します (*real_ip* とは実サーバーとの通信にディレクターが使用する IP のこと。多くの場合、**eth0** にバインドされた IP)。

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

   これにより、仮想 IP アドレス向けのすべての ARP 要求を実サーバーが無視するようになります。また、他の方法では仮想 IP を含むことになる送信 ARP 反応を変更させて、それらがサーバーの実 IP を含むようになります。VIP の ARP 要求に反応する唯一のノードは、現在アクティブな LVS ノードです。

2. これが実サーバー上で完了したら、実サーバー上で以下のコマンドを入力して ARP テーブルのエントリーを保存します。

   arptables-save > /etc/sysconfig/arptables

   systemctl enable arptables.service

   systemctl enable コマンドは、ネットワーク開始前にシステムが起動時に arptables 設定をリロードするようにします。

3. **ip addr** コマンドを使用してすべての実サーバーに仮想 IP アドレスを設定し、IP エイリアスを作成します。例を以下に示します。

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

4. ダイレクトルーティングを Keepalived に設定します。これには、***lb_kind DR*** を **keepalived.conf** ファイルに追加します。詳細は *4章Keepalived を用いたロードバランサーの初期設定* を参照してください。

3.2.2. firewalld を使用したダイレクトルーティング

firewalld でファイアウォールルールを作成することで、 ダイレクトルーティングメソッドを使用した場合の ARP 問題を回避することもできます。**firewalld** を使用してダイレクトルーティングを設定するには、VIP アドレスがシステム上に存在しなくても VIP アドレスに送信されたパケットを実サーバーが扱うように透過プロキシを作成するルールを追加する必要があります。

firewalld メソッドは **arptables** メソッドよりも設定が簡単です。単一または複数の仮想 IP アドレスがアクティブ LVS ディレクター上にのみ存在するため、このメソッドでは LVS ARP 問題も完全に回避できます。

しかし、パケットが返されるたびにオーバーヘッドが発生するため、**firewalld** メソッドは **arptables** と比較してパフォーマンスに大きく影響します。

また、**firewalld** メソッドを使用してポートを再利用することはできません。例えば、2 つの別々の Apache HTTP Server サービスは両方とも仮想 IP アドレスではなく ***INADDR_ANY*** にバインドする必要があるため、ポート 80 にバインドされた 2 つの別々の Apache HTTP Server サービスを実行することはできません。

firewalld メソッドを使用してダイレトルーティングを設定するには、すべての実サーバーで以下の手順を実行してください。

1. **firewalld** が実行されていることを確認します。

```
# systemctl start firewalld
```

firewalld がシステム起動時に開始するように有効化されていることを確認します。

```
# systemctl enable firewalld
```

2. 実サーバーに対して処理されるすべての VIP、ポート、プロトコル (TCPまたはUDP) の組み合わせに以下のコマンドを使用します。このコマンドにより、実サーバーが、与えられた VIP とポートに対するパケットを処理するようになります。

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

3. ファイアウォールルールを再読み込みし、状態情報を維持します。

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

現在の永続的設定は、新しい **firewalld** ランタイム設定と次回のシステム起動時の設定となります。

3.2.3. iptables を使用したダイレクトルーティング

iptables ファイアウォールルールを作成することで、 ダイレクトルーティングメソッドを使用した場合の ARP 問題を回避することもできます。**iptables** を使用してダイレクトルーティングを設定するには、VIP アドレスがシステム上に存在しなくても VIP アドレスに送信されたパケットを実サーバーが扱うように透過プロキシを作成するルールを追加する必要があります。

iptables メソッドは **arptables** メソッドよりも設定が簡単です。仮想 IP アドレスがアクティブ LVS ディレクター上にのみ存在するため、このメソッドでは LVS ARP 問題も完全に回避できます。

しかし、パケットが転送またはマスカレードされるたびにオーバーヘッドが発生するため、**iptables** メソッドは**arptables** と比較するしてパフォーマンスに大きく影響します。

また、**iptables** メソッドを使用してポートを再利用することはできません。例えば、2 つの別々の Apache HTTP Server サービスは両方とも仮想 IP アドレスではなく ***INADDR_ANY*** にバインドする必要があるため、ポート 80 にバインドされた 2 つの別々の Apache HTTP Server サービスを実行することはできません。

iptables メソッドを使用してダイレクトルーティングを設定するには、以下の手順を実行します。

1. 実サーバー上で、実サーバーに対応する目的の VIP、ポート、およびプロトコル (TCP または UDP) の組み合わせすべてに対して、以下のコマンドを実行します。

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

このコマンドで、実サーバーは与えられた VIP とポートが宛先となっているパケットを処理します。

2. 実サーバー上で設定を保存します。

■

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

systemctl enable コマンドは、ネットワーク開始前にシステムが起動時に iptables 設定をリロードするようにします。

3.2.4. sysctl を用いたダイレクトルーティング

sysctl インターフェースを使用して、ダイレクトルーティングの使用時に ARP の制限に対処することもできます。管理者は 2 つの **systcl** 設定を行い、実サーバーが ARP 要求で VIP をアナウンスしないようにし、VIP アドレスの ARP 要求へ応答しないようにします。この動作を有効にするには、以下のコマンドを実行します。

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

この代わりに、以下の行を **/etc/sysctl.d/arp.conf** ファイルに設定することもできます。

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

3.3. 設定の組み合わせ

使用するルーティングメソッドを決定した後、ハードウェアを接続し、設定する必要があります。

重要

LVS ルーター上のネットワークアダプターは、同じネットワークにアクセスするように設定する必要があります。たとえば、**eth0** がパブリックネットワークに接続し、**eth1** がプライベートネットワークに接続する場合、バックアップ LVS ルーター上の同じデバイスは同じネットワークに接続する必要があります。

また、起動時に最初に出現するインターフェースにリストされているゲートウェイは、ルーティングテーブルに追加され、他のインターフェースにリストされているそれ以降のゲート ウェイは無視されます。これは、実サーバーを設定する場合に特に考慮すべき重要点です。

ハードウェアをネットワークに接続したら、プライマリーおよびバックアップ LVS ルーターでネットワークインターフェースを設定します。これには、ネットワーク設定ファイルを手作業で編集する必要があります。ネットワーク設定ファイルの編集に関する詳細は、Red Hat Enterprise Linux 7 ネットワークガイドを参照してください。

3.3.1. 一般的なロードバランサーネットワーキングのヒント

Keepalived を使用してロードバランサーを設定する前に、LVS ルーターのパブリックおよびプライベートネットワークにリアル IP アドレスを設定します。各トポロジーのセクションには、ネットワークアドレスの例が記載されていますが、実際のネットワークアドレスが必要になります。以下は、ネットワークインターフェースを起動し、状態をチェックするのに便利なコマンドの一部になります。

実ネットワーキングインターフェースの起動

実ネットワークインターフェースを開くには、**root** で以下のコマンドを実行します。*N* の部分をインターフェースの番号 (**eth0** および **eth1**) に置き換えます。

ifup eth*N*

警告

ifup スクリプトで、Keepalived を使用して設定するフローティング IP アドレス (**eth0:1** または **eth1:1**) を開かないでください。代わりに、**service** または **systemctl** コマンドを使用して **keepalived** を開始してください。

実ネットワークインターフェースの停止

実ネットワークインターフェースを停止するには、**root** で以下のコマンドを実行します。*N* の部分をインターフェースの番号 (**eth0** および **eth1**) に置き換えます。

ifdown eth*N*

ネットワークインターフェースのステータスチェック

ある時点でどのネットワークインターフェースが起動しているかをチェックするには、以下のコマンドを実行します。

ip link

マシンのルーティングテーブルを表示するには、以下のコマンドを実行します。

ip route

3.3.2. ファイアウォールの要件

firewalld または **iptables** でファイアウォールを実行している場合は、VRRP トラフィックが **keepalived** ノード間を通過できるようにする必要があります。**firewalld** で VRRP トラフィックを許可するファイアウォールを設定するには、以下のコマンドを実行します。

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

ゾーンを省略すると、デフォルトのゾーンが使用されます。

ただし、**iptables** で VRRP トラフィックを許可する必要がある場合は、以下のコマンドを実行します。

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

3.4. マルチポートサービスとロードバランサー

マルチポートのロードバランサーサービスを作成する場合、LVS ルーターはどのトポロジーでも追加の設定が必要になります。HTTP (ポート 80) および HTTPS (ポート 443) などの異なる関連するプロトコルをバンドル化するファイアウォールマークを使用したり、FTP などの実際のマルチポートプロトコルを用いてロードバランサーを使用したりする場合、マルチポートサービスは人為的に作成されます。ど

ちらの場合でも、LVS ルーターはファイアウォールマークを使用して異なるポートに送信されるパケットを認識しますが、同じファイアマークが付けられているため同様に処理する必要があります。また、ファイアウォールマークを永続性と組み合わせると、永続性パラメーターによって指定された期間内に接続が発生する限り、クライアントマシンからの接続は同じホストへルーティングされます。

実サーバー上で負荷を分散するために使用されるメカニズムである IPVS は、パケットに割り当てられたファイアウォールマークを認識しますが、IPVS 自体はファイアウォールを割り当てできません。ファイアウォールマークの**割り当て**は、ネットワークパケットフィルターである **iptables** が行う必要があります。Red Hat Enterprise Linux 7 のデフォルトのファイアウォール管理ツールは、**firewalld** で、**iptables** の設定に使用できます。**iptables** を直接使用することもできます。Red Hat Enterprise Linux 7 で **iptables** を使用する方法については、Red Hat Enterprise Linux 7 セキュリティーガイドを参照してください。

3.4.1. firewalld を使用したファイアウォールマークの割り当て

管理者は **firewalld** の **firewall-cmd** ユーティリティーを使用して、特定のポートに送信されるパケットにファイアウォールマークを割り当てることができます。

firewalld が稼働していることを確認する必要がある場合は、以下を実行します。

```
# 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 を開始するには、以下を入力します。

```
# systemctl start firewalld
```

システム起動時に **firewalld** が有効な状態で開始するようにするには、以下を実行します。

```
# systemctl enable firewalld
```

ここでは、HTTP と HTTPS をバンドルする方法を例を用いて説明しますが、FTP も一般的に使用されるクラスター化されたマルチポートプロトコルです。

ファイアウォールマークを使用する場合の基本ルールとして、Keepalived でファイアウォールマークを使用する各プロトコルには、ファイアウォールマークをネットワークパケットに割り当てるための同等のファイアウォールルールが必要になります。

ネットワークパケットのフィルタールールを作成する前に、すでに他のルールが存在しないか確認します。これを行うには、シェルプロンプトを開いて、**root** でログインして以下のコマンドを実行します。

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

リッチルールが存在しない場合は、プロンプトがすぐに再表示されます。

firewalld がアクティブでリッチルールが存在する場合は、ルールのセットが表示されます。

すでに存在するルールが重要である場合は、**/etc/firewalld/zones/** の内容をチェックし、維持するルールを安全な場所にコピーしてから手順を続行します。コマンドを以下の形式で使用し、不必要なリッチルールを削除します。

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

--permanent オプションを指定すると設定が永続化されますが、次回のシステム起動後にコマンドが反映されます。設定を即座に反映する必要がある場合は、**--permanent** オプションを省略してコマンドを繰り返し実行します。

最初に設定するロードバランサーに関連するファイアウォールルールは、Keepalived サービスの VRRP トラフィックを許可します。以下のコマンドを実行します。

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

ゾーンを省略すると、デフォルトのゾーンが使用されます。

以下のルールは、ファイアウォールマーク **80** をポート 80 および 443 上でフローティング IP アドレス *n.n.n.n* が宛先となる受信トラフィックに割り当てます。

```
# 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
```

ゾーンを省略すると、デフォルトのゾーンが使用されます。

firewalld のリッチ言語コマンドの使用に関する詳細は、Red Hat Enterprise Linux 7 セキュリティーガイド を参照してください。

3.4.2. iptables を使用したファイアウォールマークの割り当て

特定ポート宛のパケットにファイアウォールマークを割り当てる場合、管理者は **iptables** を使用できます。

ここでは、HTTP と HTTPS をバンドルする方法を例を用いて説明しますが、FTP も一般的に使用されるクラスター化されたマルチポートプロトコルです。

ファイアウォールマークを使用する場合の基本ルールとして、Keepalived でファイアウォールマークを使用する各プロトコルには、ファイアウォールマークをネットワークパケットに割り当てるための同等のファイアウォールルールが必要になります。

ネットワークパケットのフィルタールールを作成する前に、すでに他のルールが存在しないか確認します。これを行うには、シェルプロンプトを開いて、**root** でログインして以下のコマンドを実行します。

/usr/sbin/service iptables status

iptables が実行されていない場合、すぐにプロンプトが再出現します。

iptables がアクティブな場合は、一連のルールが表示されます。ルールが存在する場合は、以下のコマンドを実行します。

/sbin/service iptables stop

既存のルールが重要な場合、**/etc/sysconfig/iptables** の内容を確認して、保存する価値のあるルールを安全な場所にコピーしてから継続します。

ファイアウォールルールの設定に関連する最初のロードバランサーは、Keepalived サービスの VRRP トラフィックを許可します。

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

以下のルールは、ファイアウォールマーク **80** をポート 80 および 443 上でフローティング IP アドレス *n.n.n.n* が宛先となる受信トラフィックに割り当てます。

```
/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
```

初めてルールを発行する前に、**root** としてログインし、**iptables** のモジュールをロードする必要があります。

上述の **iptables** コマンドの *n.n.n.n* は、使用中の HTTP および HTTPS 仮想サーバーのフローティング IP で置き換える必要があります。これらのコマンドは、該当するポート上の VIP が送信先となっている全トラフィックをファイアウォールマーク 80 に割り当てることと同様の効果があります。これが IPVS に認識され、適切に転送されます。

警告

上記のコマンドは即座に有効になりますが、システムを再起動すると保持されません。

3.5. FTP の設定

ファイル転送プロトコル (FTP) は旧式の複雑なマルチポートプロトコルで、ロードバランサーを使用する環境では扱いが難しくなります。扱いの難しさを理解するには、初めに FTP の仕組みで重要となる点を理解する必要があります。

3.5.1. FTP の動作

ほとんどのサーバー/クライアント関係では、クライアントマシンが特定のポート上でサーバーへ接続を開いて、サーバーがそのポートのクライアントに応答します。FTP クライアントが FTP サーバーに接続する場合、FTP 制御ポート 21 への接続を開きます。そして、その **クライアント**が FTP **サーバー**に *アクティブ* か *パッシブ* のどちらの接続を開くかを指示します。クライアントが選択した接続タイプにより、サーバーの対応方法とトランザクションが発生するポートを決定します。

データ接続は以下の 2 種類です。

アクティブ接続

アクティブ接続が確立されると、**サーバー**はポート 20 からクライアントマシン上の高い範囲のポートにクライアントへデータ接続を開きます。サーバーからのすべてのデータは、この接続を通じて送信されます。

パッシブ接続

パッシブ接続が確立されると、**クライアント**は FTP サーバーに対してパッシブ接続ポートを確立するように依頼します。これは 10,000 より高いポートになります。するとサーバーは、この特定の

セッション用に高い数値のポートをバインドして、このポート番号をクライアントに中継します。クライアントは、データ接続のために新規にバインドされたポートを開きます。クライアントが作成するデータ要求それぞれ、別個のデータ接続となります。最近の FTP クライアントのほとんどは、 サーバーからデータを要求する場合、パッシブ接続を試みます。

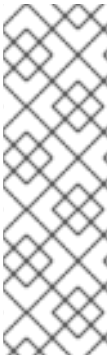
注記

接続タイプを決定するのは、サーバーではなく **クライアント** です。つまり、効果的に FTP をクラスター化するには、アクティブ接続とパッシブ接続の両方を処理するように LVS ルーターを設定する必要があることになります。

HTP のクライアントとサーバーの関係によって、Keepalived が認識しない多くのポートが開かれる可能性があります。

3.5.2. ロードバランサーのルーティングへの影響

IPVS パケット転送は、それをベースにしたクラスターへの接続とそのクラスターからの接続のみを許可し、そのポート番号やファイアウォールマークを認識します。クラスター外のクライアントが IPVS で処理するように設定されていないポートを開こうとした場合、接続は切断されます。同様に、実サーバーが IPVS が認識できないポート上でインターネット接続を開こうとした場合も、接続は切断されます。つまり、インターネット上の FTP クライアントからの**すべての**接続は、それらに割り当てられているファイアウォールマークと同じである**必要があり**、FTP サーバーからの全接続は、ネットワークパケットのフィルタリングルールを使用して正常にインターネットに転送される**必要がある**ことを意味します。



注記

パッシブ FTP 接続を有効にするには、**ip_vs_ftp** カーネルモジュールを読み込む必要があります。以下のコマンドを管理者ユーザーとしてシェルで実行し、このモジュールを読み込み、再起動してモジュールが読み込まれることを確認します。

```
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. ネットワークパケットフィルタルールの作成

FTP サービスの **iptables** ルールを割り当てる前に、「マルチポートサービスとロードバランサー」 で既存のネットワークパケットフィルタールールをチェックするためのマルチポートサービスおよびテクニックに関する情報を確認してください。

以下のルールは、ファイアウォールマーク **21** をFTP トラフィックへ割り当てます。

3.5.3.1. アクティブ接続のルール

アクティブ接続のルールは、ポート **20** (FTP データポート) 上で **内部** のフローティング IP アドレス宛の接続を許可および転送するようカーネルに指示します。

以下の **iptables** コマンドにより、LVS ルーターは IPVS が認識していない実サーバーからの外向けの接続を受け付けることが可能になります。

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

この **iptables** コマンドの *n.n.n* は、**keepalived.conf** ファイルの ***virtual_server*** セクションで定義され、NAT インターフェースの内部ネットワークインターフェースに割り当てられるフローティング IP の最初の 3 つの値に置き換える必要があります。

3.5.3.2. パッシブ接続のルール

パッシブ接続のルールは、10,000 から 20,000 までの広い範囲のポートで、インターネットからフローティング IP アドレスへの接続に適切なファイアウォールマークを割り当てます。

警告

パッシブ接続でポート範囲を制限している場合、FTP サーバーである **vsftpd** を設定し、同じポートの範囲を使用するようにする必要があります。これには、以下の行を **/etc/vsftpd.conf** に追加します。

pasv_min_port=10000

pasv_max_port=20000

実際の FTP サーバーアドレスを上書きする **pasv_address** の設定は使用しないでください。LVS により仮想 IP アドレスに更新されるためです。

他の FTP サーバーの設定については、個別のドキュメンテーションを参照してください。

この範囲はほとんどの状況では十分なものです。しかし、以下のコマンド内の **10000:20000** を **1024:65535** に変更することで、利用可能な非保護ポートすべてを含めることができます。

以下の **iptables** コマンドは、適切なポート上のフローティング IP 宛のトラフィックにファイアウォールマーク **21** を割り当てます。これにより、IPVS で認識され、適切に転送されます。

/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

この **iptables** コマンドの *n.n.n.n* は、**keepalived.conf** ファイルの ***virtual_server*** サブセクションで定義される FTP 仮想サーバーのフローティング IP アドレスに置き換える必要があります。

上記のコマンドは即座に有効になりますが、保存されていないとシステムの再起動後に保持されません。変更を保存するには、以下のコマンドを実行します。

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

システムの起動時に **iptables** サービスが開始されるようにするには、以下のコマンドを実行します。

```
# systemctl enable iptables
```

再起動時も変更が維持されていることを確認するには、以下のコマンドを実行して変更を確認してください。

```
# systemctl restart iptables
```

3.6. ネットワークパケットフィルター設定の保存

ユーザーの状況に応じた適切なネットワークパケットフィルターを設定した後は、 その設定を保存して再起動後に復元するようにします。**iptables** には以下のコマンドを実行します。

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

システムの起動時に **iptables** サービスが開始されるようにするには、以下のコマンドを実行します。

```
# systemctl enable iptables
```

再起動時も変更が維持されていることを確認するには、以下のコマンドを実行して変更を確認してください。

```
# systemctl restart iptables
```

Red Hat Enterprise Linux 7 で **iptables** を使用するための詳細は、Red Hat Enterprise Linux 7 セキュリティーガイド を参照してください。

3.7. 「パケット転送および非ローカルバインディングの有効化」

Keepalived サービスが適切にネットワークパケットを実サーバーへ転送するには、各ルーターノードのカーネルで IP 転送が有効になっている必要があります。 **root** としてログインし、**/etc/sysctl.conf** にある行 **net.ipv4.ip_forward = 0** を以下に変更します。

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

システムを再起動すると変更が反映されます。

HAProxy と Keepalived で同時に負荷分散を行うには、ローカルシステムのデバイスに割り当てられていない*非ローカル*の IP アドレスへバインドする機能が必要になります。この機能により、フェイルオーバーの発生時に稼働しているロードバランサーインスタンスがローカルでない IP アドレスへバインドできるようになります。

この機能を有効にするには、**/etc/sysctl.conf** にある行 **net.ipv4.ip_nonlocal_bind** を以下に変更します。

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

システムを再起動すると変更が反映されます。

IP 転送が有効になっているのを確認するには、**root** で以下のコマンドを実行します。

/usr/sbin/sysctl net.ipv4.ip_forward

非ローカルバインディングが有効になっていることを確認するには、**root** で以下のコマンドを実行します。

/usr/sbin/sysctl net.ipv4.ip_nonlocal_bind

設定が有効になっていれば、上記の両方のコマンドが **1** を返します。

3.8. 実サーバーでサービスを設定する

実サーバーが Red Hat Enterprise Linux システムであれば、適切なサーバーデーモンが起動時にアクティベートするように設定します。これらのデーモンには、Web サービスの **httpd** もしくは FTP または Telnet サービスの **xinetd** を含めることができます。

また、実サーバーに遠隔からもアクセスできると便利なため **sshd** デーモンもインストールして実行しておいてください。

第4章 KEEPALIVED を用いたロードバランサーの初期設定

Load Balancer パッケージをインストールしたら、基本手順に従って Keepalived で使用する LVS ルーターと実サーバーを設定する必要があります。本章では、最初の手順を説明します。

4.1. 基本的な KEEPALIVED の設定

この基本的な例では、2 つのシステムがロードバランサーとして設定されます。LB1 (アクティブ) および LB2 (バックアップ) は、仮想 IP アドレス 10.0.0.1 を共有し、192.168.1.20 から 192.168.1.24 までの実際の IP アドレスで **httpd** を実行する 4 つの Web サーバーのプールに対する要求をルーティングします。各ロードバランサーには 2 つのインターフェースがあり (**eth0** および **eth1**)、その 1 つは外部インターネットを処理し、もう 1 つは要求を実サーバーへルーティングします。ロードバランシングアルゴリズムにはラウンドロビンアルゴリズムが使用され、ルーティングの方法はネットワークアドレス変換になります。

4.1.1. keapalived.conf ファイルの作成

Keepalived は、ロードバランサーとして設定された各システムの **keepalived.conf** ファイルで設定します。「基本的な Keepalived の設定」 の例のようなロードバランサートポロジーを作成するには、テキストエディターで、アクティブのロードバランサーとバックアップロードバランサー、LB1 と LB2 の両方で **keepalived.conf** を開きます。以下に例を示します。

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

「基本的な Keepalived の設定」 で説明されているような構成で負荷分散されている基本的なシステムには以下のコードセクションで示すような **keepalived.conf** ファイルがあります。この例では、**keepalived.conf** ファイルがアクティブルーターとバックアップルーターの両方で同じものになります。ただし、「VRRP インスタンス」 で説明されている VRRP インスタンスは例外です。

4.1.1.1. グローバル定義

keepalived.conf ファイルのグローバル定義セクションを使用すると管理者はロードバランサーに変化が生じた場合の通知詳細を指定することができます。グローバル定義は任意で、Keepalived の設定には必要ありません。**keepalived.conf** ファイルのこの項は、LB1 と LB2 の両方で同じです。

```
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 はロードバランサーの管理者、***notification_email_from*** はロードバランサーの状態の変化を送信するアドレスです。SMTP 固有の設定で通知の送信元となるメールサーバーを指定します。

4.1.1.2. VRRP インスタンス

以下の例では、マスタールーターとバックアップルーターの **keeplalived.conf** ファイルの ***vrrp_sync_group*** スタンザを示しています。**state** と **priority** 値はこれら 2 つのシステム間で異なることに注意してください。

以下の例では、マスタールーターの LB1 の **keepalived.conf** ファイルの ***vrrp_sync_group*** スタンザを示しています。

```
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
   }
}
```

以下の例では、バックアップルーターである LB2 の **keepalived.conf** ファイルの ***vrrp_sync_group*** スタンザを示しています。

```
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
   }
}
```

これら例では、 ***vrrp_sync_group*** スタンザでは、状態が変化 (フェールオーバーなど) しても VRRP グループは変化しないことが定義されています。インターネットと通信を行う外部インターフェースに対するインスタンス (RH_EXT) の他、内部インターフェースに対するインスタンス (RH_INT) も定義されています。

vrrp_instance の行は、仮想 IP インスタンスを作成する VRRP サービスデーモンの仮想インターフェース設定の詳細になります。**state MASTER** はアクティブサーバーを指定し、**state BACKUP** はバックアップサーバーを指定します。

interface パラメーターは、この仮想 IP インスタンスに物理的なインターフェイス名を割り当てます。

virtual_router_id は Virtual Router インスタンスの数値識別子です。これは、この Virtual Router に加わっているすべての LVS Router システムで同じでなければなりません。また、同じネットワークインターフェイス上で実行している複数の **keepalived** のインスタンスを差別化するのに使用されます。

priority は、割り当てられたインターフェースがフェイルオーバーの発生時に引き継ぐ順番を指定します。数値が高いほど優先度が高くなります。この優先値は 0 から 255 の範囲内である必要があり、**state MASTER** として設定されたロードバランシングサーバーの優先値には、**state BACKUP** として設定されたサーバーの優先値よりも大きい値を設定する必要があります。

authentication のブロックは、サーバーをフェイルオーバーの同期に対して認証するための認証タイプ (**auth_type**) およびパスワード (**auth_pass**) を指定します。**PASS** はパスワード認証を指定します。Keepalived は、接続の整合性を目的とした認証ヘッダーである **AH** もサポートします。

最後に、**virtual_ipaddress** オプションではインターフェースの仮想 IP アドレスを指定します。

4.1.1.3. 仮想サーバーの定義

keepalived.conf ファイルの Virtual Server 定義セクションは、LB1 と LB2 の両方で同じです。

```
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
        }
    }

}
```

最初に IP アドレスを指定して **virtual_server** を設定します。次に、**delay_loop** に健全性チェックの間隔を秒単位で設定します。**lb_algo** オプションには可用性に使用されるアルゴリズムの種類を指定します。この例では、ラウンドロビンである **rr** が指定されています。**lb_algo** に指定可能な値は 表4.1「仮想サーバーの lv_algo の値」 を参照してください。**lb_kind** オプションにはルーティングの方法を指定します。この例では、ネットワークアドレス変換 (or **nat**) が指定されています。

仮想サーバーの詳細を設定したら、**real_server** オプションを設定します。ここでもまず IP アドレスを最初に指定します。**TCP_CHECK** スタンザでは TCP を使って 実サーバーの可用性をチェックします。**connect_timeout** ではタイムアウトが発生するまでの時間を秒単位で設定します。

注記

ロードバランサーまたは実サーバーのいずれかからの仮想 IP アクセスはサポートされていません。また、実サーバーと同じマシン上でのロードバランサーの設定もサポートされていません。

表4.1 仮想サーバーの lv_algo の値

アルゴリズム名	lv_algo の値
ラウンドロビン (Round-Robin)	**rr**
加重ラウンドロビン (Weighted Round-Robin)	**wrr**

アルゴリズム名	lv_algo の値
最小接続 (Least-Connection)	**lc**
加重最小接続 (Weighted Least-Connection)	**wlc**
ローカリティーベースの最小接続 (Locality-Based Least-Connection)	**lblc**
複製をともなうローカリティベースの最小接続スケジューリング (Locality-Based Least-Connection Scheduling with Replication)	**lblcr**
宛先ハッシュ (Destination Hash)	**dh**
ソースハッシュ (Source Hash)	**sh**
ソースの予期される遅延 (Source Expected Delay)	**sed**
キューに置かない (Never Queue)	**nq**

4.2. KEEPALIVED ダイレクトルーティング設定

Keepalived のダイレクトルーティング設定は NAT での設定と似ています。次の例ではポート 80 で HTTP を実行している実サーバーのグループに負荷分散を提供するよう Keepalived が設定されています。ダイレクトルーティングを設定する場合は、***lb_kind*** パラメーターを **DR** に変更します。他の設定オプションの説明は 「基本的な Keepalived の設定」 を参照してください。

以下の例では、ダイレクトルーティングを使用する Keepalived 設定のアクティブサーバーの **keepalived.conf** を示しています。

```
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
        }
    }
}
```

以下の例では、ダイレクトルーティングを使用するバックアップサーバーの **keepalived.conf** ファイルを示しています。**state** と **priority** 値は、アクティブサーバーの **keepalived.conf** と異なることに注意してください。

```
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. サービスの開始

ロードバランサー設定のサーバー上で以下のコマンドを実行してサービスを起動します。

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

再起動後も Keepalived サービスを維持する場合は、ロードバランサーの設定のサーバー上で以下のコマンドを実行します。

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

第5章 HAPROXY の設定

本章では、可用性の高い環境に HAProxy を導入する際の一般的な設定オプションに焦点を置きながら基本的なセットアップの構成について説明していきます。

HAProxy は、負荷分散を行うための独自のスケジューリングアルゴリズムのセットを持っています。これらのアルゴリズムは 「HAProxy のスケジューリングアルゴリズム」 に記載されています。

HAProxy の設定を行う場合は **/etc/haproxy/haproxy.cfg** ファイルを編集します。

HAProxy を使ったロードバランサーの設定は 5 つのセクションで構成されます。

- 「グローバル設定」
- 4 つのサブセクションで構成されるプロキシーのセクション
  - 「デフォルトの設定」 のセッティング
  - 「Frontend 設定」 のセッティング
  - 「バックエンドの設定」 のセッティング

5.1. HAPROXY のスケジューリングアルゴリズム

負荷分散用の HAProxy スケジューリングアルゴリズムは、**/etc/haproxy/haproxy.cfg** 設定ファイルの ***backend*** セクションにある ***balance*** パラメーターで編集できます。HAProxy は複数のバックエンドを持つ設定をサポートし、各バックエンドにスケジューリングアルゴリズムを設定できます。

ラウンドロビン (*roundrobin*)

要求を順番に実サーバーのプールで分配します。このアルゴリズムを使用すると、容量や負荷に関係なく実サーバーはすべて同等に扱われます。このスケジューリングモデルはラウンドロビン DNS と似ていますが、ホストベースではなくネットワークベースであるため、粒度がより細かくなります。さらに、ロードバランサーのラウンドロビンスケジューリングでは、キャッシュされた DNS クエリーが原因で負荷が不均等になることはありません。しかし、HAProxy ではこのスケジューラーを使用するとサーバーの加重値をオンザフライで設定できるため、バックエンドごとにアクティブなサーバーの数が 4095 に制限されます。

静的ラウンドロビン (*static-rr*)

ラウンドロビンと同様に、要求を順番に実サーバーのプールで分配しますが、サーバーの加重値を動的に設定できません。しかし、サーバーの加重値が静的であるため、アクティブなサーバーの数はバックエンドごとに制限されません。

最小接続 (*leastconn*)

アクティブな接続の数が少ない実サーバーにより多くの要求を分配します。セッションや接続の長さが異なる動的な環境に適したスケジューラーです。また、このスケジューラーを使用すると管理者は加重値をオンザフライで調整できるため、異なる容量のサーバーが複数ある環境にも適しています。

ソース (*source*)

要求するソース IP アドレスをハッシュ化し、稼働するすべてのサーバーの加重値で割って要求を取得するサーバーを決定し、要求をサーバーに分配します。すべてのサーバーが稼働中である場合、ソース IP 要求は一貫して同じ実サーバーによって処理されます。稼働中のサーバーの数や加重値に変更があった場合、ハッシュまたは加重値の結果が変わるため、セッションが別のサーバーに移動される可能性があります。

URI (*uri*)

URI 全体 (または URI の設定可能な部分) をハッシュ化し、稼働するすべてのサーバーの加重値で割ってサーバーへ要求を分配します。アクティブなサーバーがすべて稼働中である場合、宛先 IP 要求は一貫して同じ実サーバーによって処理されます。このスケジューラーを追加設定するには、URI のディレクトリー部分の最初にある文字列の長さでハッシュ化の結果を算出し、URI のディレクトリーの深さ (URI のスラッシュによる) でハッシュ化の結果を算出します。

URL パラメーター (*url_param*)

ソース URL 要求の特定のパラメーター文字列を検索し、ハッシュの計算を稼働中のすべてのサーバーの加重値で割って、サーバーへの要求を分配します。URL にパラメーターがない場合、スケジューラーはラウンドロビンスケジューリングをデフォルトとして使用します。POST パラメーターを基にした修飾子が使用されたり、ハッシュ化の結果を算出する前に管理者が特定パラメーターの重さに割り当てる最大オクテットの数を基にした待機制限が使用されることがあります。

ヘッダー名 (*hdr*)

各リソース HTTP 要求の特定のヘッダー名をチェックし、ハッシュの計算を稼働中のサーバーすべての加重値で割って、サーバーへ要求を分配します。ヘッダーがない場合は、スケジューラーはデフォルトでラウンドロビンスケジューリングを使用します。

RDP クッキー (*rdp-cookie*)

各 TCP 要求に対して RDP クッキーを検索し、ハッシュの計算を稼働中のサーバーすべての加重値で割って、サーバーへ要求を分配します。ヘッダーがない場合、スケジューラーはデフォルトでラウンドロビンスケジューリングを使用します。この方法はセッションの整合性を維持するため、永続化に適しています。

5.2. グローバル設定

global セッティングでは HAProxy を実行するサーバーすべてに適用するパラメーターを設定します。一般的な ***global*** セッティングを以下に示します。

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

上記の設定では、サービスはすべてのエントリーをローカルの **syslog** サーバーに ***log*** するよう設定します。デフォルトでは **/var/log/syslog** またはユーザーが指定する場所になる場合があります。

maxconn パラメーターではサービスの並列接続の最大数を指定しています。デフォルトの最大数は 2000 です。

user と ***group*** のパラメーターは **haproxy** プロセスが属するユーザー名とグループ名を指定します。

daemon パラメーターでは **haproxy** がバックグラウンドプロセスとして実行されるよう指定しています。

5.3. デフォルトの設定

default のセッティングでは設定内の全プロキシサブセクションに適用されるパラメーターを設定します (***frontend***、***backend***、***listen***)。一般的な ***default*** セクションを以下に示します。

注記

proxy サブセクション (***frontend***、***backend***、***listen***) 内で設定するパラメーターはすべて ***default*** 内のパラメーター値より優先されます。

```
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 では HAProxy インスタンスのプロトコルを指定しています。***http*** モードを使用するとソースの要求は HTTP に応じて実サーバーに接続されるため、web サーバーの負荷分散を行う場合に理想的です。他のアプリケーションの場合は ***tcp*** モードを使用してください。

log ではログのエントリーが書き込まれる **syslog** ファシリティとログアドレスを指定しています。***global*** の値は ***global*** セクションの ***log*** パラメーターに指定されている HAProxy インスタンスを指しています。

option httplog は HTTP 要求、セッション状態、接続数、ソースアドレス、他の値内での接続タイマーなど、HTTP セッションの各種の値のログ記録を有効にしています。

option dontlognull は null 接続のログ記録を無効にしています。つまり、HAProxy はデータが転送されなかった場合の接続はログ記録しないということです。この設定はインターネット経由の web アプリケーションを使用するなどの環境、つまり脆弱性を狙ってポートスキャンを開くなど悪意のある動きを null 接続で検出できるような場合には推奨できません。

retries では最初の接続試行に失敗した後、実サーバーに再接続の要求を何回行わせるかを指定しています。

timeout の値は、指定の要求、接続、または応答の非アクティブな時間の長さを指定します。これらの値は、明示的な指定がある場合を除きミリ秒単位で指定されます。他の単位で指定する場合は、その単位を数値の後に追加します。サポートされる単位は us (マイクロ秒)、ms (ミリ秒)、s (秒)、m (分)、h (時)、および d (日) です。 ***http-request 10s*** を指定すると、クライアントからの完全な HTTP 要求を 10 秒間待ちます。 ***queue 1m*** を指定すると、1 分間待機した後に接続が切断され、クライアントは 503 エラー (「Service Unavailable 」サービス利用不可エラー) を受信します。***connect 10s*** を指定すると、サーバーへ正常に接続できるまで 10 秒待機します。***client 1m*** は、クライアントが非アクティブな状態 (データの送受信がない状態) でいられる時間を 1 分に指定します。 ***server 1m*** を指定すると、サーバーでデータの送受信が 1 分間行われないと、タイムアウトが発生します。

5.4. FRONTEND 設定

frontend のセッティングではクライアントの接続要求を受けるサーバーのリスニングソケットを設定します。一般的な ***frontend*** の HAProxy 設定を以下に示します。

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

main という名前の ***frontend*** は、***bind*** パラメータを使用して IP アドレス 192.168.0.10 に設定され、ポート 80 でリッスンされます。接続後、***use backend*** は ***app*** バックエンドにすべてのセッションが接続するよう指定します。

5.5. バックエンドの設定

backend のセッティングでは実サーバーの IP アドレスおよびロードバランサーのスケジューリングアルゴリズムを指定します。一般的な ***backend*** セクションを以下に示します。

```
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
```

バックエンドサーバーの名前は ***app*** です。***balance*** は使用されるロードバランサースケジューリングアルゴリズムを指定します。この例ではラウンドロビン (***roundrobin***) が指定されていますが、HAProxy がサポートする他のスケジューラーを指定することもできます。HAProxy でのスケジューラーの設定については、「HAProxy のスケジューリングアルゴリズム」 を参照してください。

server の行はバックエンドで利用できるサーバーを指定しています。***app1*** から ***app4*** は、内部的に各サーバーに割り当てられた名前になります。ログファイルは名前でサーバーメッセージを指定します。アドレスは割り当てられた IP アドレスです。IP アドレスのコロンの後にある値は、そのサーバーで接続が確立されるポート番号です。***check*** オプションは、サーバーの定期的な健全性チェックを指定し、サーバーが利用可能であるようにし、データの送受信やセッション要求の受信を行えるようにします。サーバー app3 は、健全性チェックの間隔が 2 秒 (***inter 2s***) に設定され、app3 が健全であると判断されるチェックの合格数 (***rise 4***) やチェックに失敗したと判断される連続失敗数 (***fall 3***) も設定されています。

5.6. HAPROXY の開始

HAProxy サービスを開始するには、以下のコマンドを入力します。

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

再起動後も HAProxy サービスを維持する場合は次のコマンドを入力します。

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

5.7. RSYSLOG への HAPROXY メッセージのロギング

/dev/log ソケットに書き込むことで、HAProxy メッセージを **rsyslog** に記録するようにシステムを設定できます。また、TCP ループバックアドレスを対象にすることもできますが、パフォーマンスが遅くなります。

以下の手順では、メッセージを **rsyslog** に記録するように HAProxy を設定します。

1. HAProxy 設定ファイルの **global** セクションで、**log** ディレクティブを使用して **/dev/log** ソケットを対象にします。

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

2. **frontend** プロキシー、**backend** プロキシー、および **listen** プロキシーを更新して、HAProxy 設定ファイルの **global** セクションに設定した **rsyslog** サービスにメッセージを送信します。これを行うには、**log global** ディレクティブを HAProxy 設定ファイルの **defaults** セクションに、**log global** ディレクティブを追加します。

```
defaults
   log global
   option httplog
```

3. **chrooted** 環境で HAProxy を実行している場合、または **chroot** 設定ディレクティブを使用して HAProxy に **chroot** ディレクトリーを作成する場合は、その **chroot** ディレクトリー内で利用できるようにする必要があります。これを行うには、**rsyslog** 設定を修正して、**chroot** ファイルシステムで新しい待機 (リッスン) ソケットを作成します。これを行うには、以下の行を **rsyslog** 設定ファイルに追加します。

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

4. HAProxy ログメッセージの表示内容や場所をカスタマイズするには、『システム管理者のガイド』のRsyslog の基本設定に従って **rsyslog** フィルターを使用します。

付録A 設定例: HAPROXY および KEEPALIVED を用いた CEPH OBJECT GATEWAY サーバーの負荷分散

本付録では、Ceph クラスターにおける HAProxy および Keepalived の設定例を説明します。Ceph Object Gateway を使用すると、オブジェクトゲートウェイの多くのインスタンスを 1 つのゾーンに割り当てできるため、負荷の増加に合わせて拡張することができます。各オブジェクトゲートウェイは独自の IP アドレスを持つため、HAProxy および keepalived を使用して Ceph Object Gateway サーバー全体で負荷を分散できます。

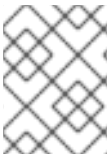
この設定では、HAProxy は Ceph Object Gateway サーバー全体で負荷分散を行います。 Keepalived は、Ceph Object Gateway サーバーの仮想 IP アドレスの管理および HAProxy の監視に使用されます。

HAProxy および keepalived では、HAProxy サーバーで HTTPS を終了するユースケースがあります。Red Hat Ceph Storage (RHCS) 1.3.x は Civetweb を使用しますが、RHCS 1.3.x の実装は HTTPS をサポートしません。HAProxy サーバーを使用して HAProxy サーバーで HTTPS を終了でき、HAProxy サーバーと Civetweb ゲートウェイインスタンスの間で HTTP を使用できます。この例には、手順の一部としてこの設定が含まれます。

A.1. 前提条件

Ceph Object Gateway で HAProxy を設定するには、以下が必要になります。

- 稼働中の Ceph クラスター
- ポート 80 で実行されるよう設定された同じゾーン内の Ceph Object Gateway サーバー 2 台以上
- HAProxy および keepalived のサーバー 2 台以上



注記

この手順では、2 台以上の Ceph Object Gateway サーバーが実行され、ポート 80 上でテストスクリプトを実行すると有効な応答が返されることを前提とします。

A.2. HAPROXY ノードの準備

次の設定では、 **haproxy** と **haproxy2** という名前の 2 つの HAProxy ノードがあり、 **rgw1** と **rgw2** という名前の 2 つの Ceph Object Gateway サーバーがあることを前提とします。希望の命名規則を使用できます。以下の手順を 2 つの HAProxy ノードで実行します。

1. Red Hat Enterprise Linux 7 をインストールします。
2. ノードを登録します。

```
# subscription-manager register
```

3. Red Hat Enterprise Linux 7 サーバーリポジトリーを有効にします。

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

4. サーバーを更新します。

```
# yum update -y
```

5. 必要に応じて管理ツール (**wget**、**vim** など) をインストールします。

6. ポート 80 を開きます。

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

7. HTTPS 用にポート 443 を開きます。

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

A.3. KEEPALIVED のインストールおよび設定

以下の手順を 2 つの HAProxy ノードで実行します。

1. keepalived をインストールします。

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

2. keepalived を設定します。

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

以下の設定には、HAProxy のプロセスをチェックするスクリプトがあります。インスタンスは **eth0** をネットワークインターフェースとして使用し、**haproxy** をマスターサーバー、**haproxy2** をバックアップサーバーとして設定します。また、インスタンスは仮想 IP アドレス 192.168.0.100 を割り当てます。

```
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. keepalived を有効化し、開始します。

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

A.4. HAPROXY のインストールおよび設定

以下の手順を 2 つの HAProxy ノードで実行します。

1. **haproxy** をインストールします。

```
# yum install haproxy
```

2. SELinux および HTTP に対して **haproxy** を設定します。

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

以下の行を追加します。

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

root ユーザーとして、適切な SELinux コンテキストとファイルパーミッションを **haproxy-http.xml** ファイルに割り当てます。

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

3. HTTPS を使用する場合は、SELinux および HTTPS に対して **haproxy** を設定します。

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

以下の行を追加します。

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

root ユーザーとして、適切な SELinux コンテキストとファイルパーミッションを **haproxy-https.xml** ファイルに割り当てます。

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

4. HTTPS を使用する場合は、SSL のキーを生成します。証明書がない場合は、自己署名証明書を使用できます。キーの生成や自己署名証明書に関する詳細は、Red Hat Enterprise Linux の『システム管理者のガイド』を参照してください。

   最後に、証明書と鍵を PEM ファイルに格納します。

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

5. HAProxy を設定します。

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

   haproxy.cfg の **global** および **defaults** セクションは変更されません。以下の例のように、**defaults** セクションの後に **frontend** および **backend** セクションを設定する必要があります。

   ```
   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. **haproxy** を有効にし、開始します。

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

A.5. HAPROXY 設定のテスト

HAProxy ノードで、**keepalived** 設定の仮想 IP アドレスが表示されることを確認します。

```
$ ip addr show
```

calamari ノードで、ロードバランサー設定を使用してゲートウェイノードに接続できるか確認します。例を以下に示します。

```
$ wget haproxy
```

上記のコマンドの結果は以下のコマンドと同じになるはずです。

```
$ wget rgw1
```

以下の内容を持つ **index.html** ファイルが取得できれば、設定は適切に動作しています。

```
<?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>
```

付録B 改訂履歴

改訂 4.1-1 Wed Aug 7 2019 Steven Levine
7.7 GA 公開用ドキュメントの準備

改訂 3.1-2 Thu Oct 4 2018 Steven Levine
7.6 GA 公開用ドキュメントの準備

改訂 2.1-1 Thu Mar 15 2018 Steven Levine
7.5 GA 公開用ドキュメントの準備

改訂 2.1-0 Thu Dec 14 2017 Steven Levine
7.5 ベータ版公開用ドキュメントの準備

改訂 0.6-5 Wed Nov 22 2017 Steven Levine
7.4 のバージョンを更新

改訂 0.6-3 Thu Jul 27 2017 Steven Levine
7.4 GA 公開用ドキュメントバージョン

改訂 0.6-1 Wed May 10 2017 Steven Levine
7.4 ベータ版公開用ドキュメントの準備

改訂 0.5-9 Mon Dec 5 2016 Steven Levine
7.3 のバージョンを更新

改訂 0.5-7 Mon Oct 17 2016 Steven Levine
7.3 GA 公開用バージョン

改訂 0.5-6 Thu Aug 18 2016 Steven Levine
7.3 Beta 公開用ドキュメントの準備

改訂 0.3-2 Mon Nov 9 2015 Steven Levine
7.2 GA 公開用ドキュメントの準備。

改訂 0.3-0 Wed Aug 19 2015 Steven Levine
7.2 ベータ公開用ドキュメントの準備

改訂 0.2-6 Mon Feb 16 2015 Steven Levine
7.1 GA リリース向けのバージョン

改訂 0.2-5 Thu Dec 11 2014 Steven Levine
7.1 ベータリリース向けバージョン

改訂 0.2-4 Thu Dec 04 2014 Steven Levine
7.1 ベータリリース向けバージョン

改訂 0.1-12 Tue Jun 03 2014 John Ha
7.0 GA リリース向けバージョン

改訂 0.1-6 Mon Jun 13 2013 John Ha
Red Hat Enterprise Linux 7 のベータ向けビルド

改訂 0.1-1 Wed Jan 16 2013 John Ha
Red Hat Enterprise Linux 6 バージョンのドキュメントからブランチ作成

索引

シンボル

N