コンテナクラスタ用ポータブルロードバランサの検討

高橋公俊 [1]　　合田憲人 [2,1]

1. はじめに

Linux コンテナのクラスタからなる WEB システムが注目を集めている。Linux コンテナは可搬性が高く、BCP(Business continuity planning)や DR(Disaster Recovery)の目的で WEB システムを異なるクラウドプロバイダのクラウド基盤間でマイグレーションすることが容易であるという利点がある。

しかし既存のコンテナ管理システム Kubernetes[1]を調査したところ、異なるクラウドプロバイダ間でマイグレーションを実現するためには、ロードバランサの仕組みに課題があることがわかった。具体的には、Kubernetes は外部のロードバランサに依存しており、コンテナクラスタの起動と同時にクラウドプロバイダが提供する API を通してロードバランサのセットアップを行う。ところが、ロードバランサの仕組みや API はクラウドプロバイダによって異なり、今の所一部の主要なクラウドプロバイダでしかロードバランサを使うことができない。また、オンプレミスデータセンタ用のロードバランサ製品は多岐に渡り、その多くを Kubernetes から API で制御することは非常に困難である。本稿ではこの課題を解決するためにロードバランサ自体をコンテナとして実装する方法について検討する。

2. Kubernetes におけるロードバランサの課題

Fig.1 に Kubernetes システムの構成図を示す。Kubernetes システムは Master と複数の Node で構成される。Master が kubectl コマンドによるリクエストを受けると、各ノードで Pod が作成される。Pod は IP アドレスなどのリソースを共有する複数のコンテナをまとめたもので、Kubernetes では Pod 単位でコンテナが実行される。

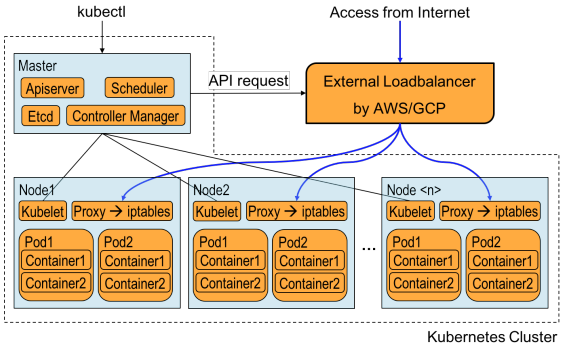

Fig.1 Kubernetes システム構成図

Kubernetes でのロードバランサのセットアップは、Master からの API リクエストを通して行われる。セットアップされたロードバランサはインターネットからのアクセスを全ての Node に均等に振り分け、各 Node でそれらのアクセスをさらに適切な Pod に DNAT[2]を用いて接続している。 以上が Kubernetes における外部ロードバランサの利用形態であるが、この方法には次のような問題がある。(1)ロードバランサの仕組みや API はクラウドプロバイダによって異なるため、Kubernetes にサポートされた非常に限られたクラウドプロバイダ以外はこの仕組みを使うことができない。(2)オンプレミスデータセンタで使われるロードバランサ製品は多岐に渡り、そもそもそれらの多くをサポートすることは非常に困難である。(3)Pod への DNAT は iptables を利用しているため、同時にデプロイされる WEB システムの数が増えると、設定が複雑になりデバッグが困難になる。(4)一旦全ての Node にアクセスを均等に分散し、そこから目的の Pod に DNAT を用いて接続してるため、アクセス経路が複雑になり障害ポイントの把握が困難である。

3. 本研究での提案手法

Fig.2 に本稿が提案するロードバランサを用いた場合のシステム構成を示す。

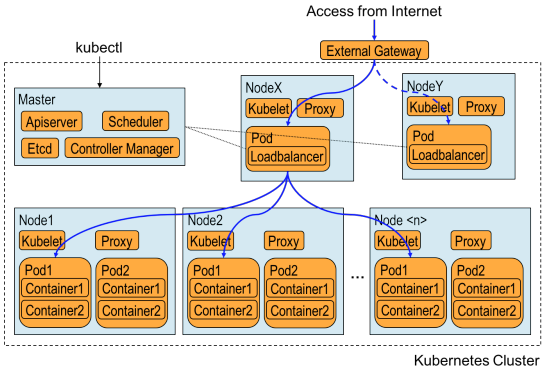

Fig.2 ポータブルロードバランサシステム構成図

提案のロードバランサは次のような特徴を持っている。 (1)ロードバランサ自体が Kubernetes の Pod として起動される。(2)ロードバランサ用コンテナは冗長化されており、Active-Backup 構成をとる。(3)負荷分散先の Pod の情報を Master から取得し、動的に分散先に加えたり分散先から外したりすることができる。

このような構成をとることで、既存の Kubernetes システムでのロードバランサの問題を解決することができる。すなわち、ロードバランサ自体をコンテナ化しているので、Kubernetes による API サポートの如何に関わらず、オンプレミス環境でもクラウドプロバイダでも同じ仕組みのロードバランサを利用することができる。また iptables DNAT に依存しないためデバッグの複雑さが無く、外部からのアクセス経路も単純で、通信障害発生時にも、障害箇所を容易に特定することができる。

4. 実装検討

今回我々は、提案方式の実現可能性を明らかにするために、オンプレミス KVM 仮想マシン[3]と AWS EC2 上に Fig. 2 の Kubernetes システムを構築し、ロードバランサ実装方法について検討を行った。

4.1 KVM 仮想環境

Fig.3 に KVM 環境でのロードバランサが稼働する Node のネットワーク構成図を示す。

1 総合研究大学院大学、2 国立情報学研究所

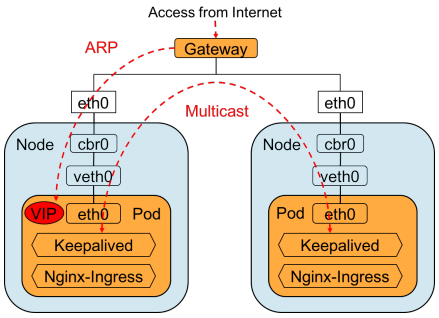

Fig.3 KVM 環境ロードバランサネットワーク構成

外部からゲートウェイに到達した VIP 宛パケットは、MAC アドレスを ARP 解決したのちに、イーサネットフレームとしてロードバランサ用 Pod に送られる。ロードバランサ用 Pod では Nginx-Ingress[4]コンテナが稼働しており、そこから複数の Pod に HTTP パケットを送信することで負荷分散を実現している。ロードバランサ用 Pod では、keepalived[5]も稼働しており、他のロードバランサ Pod との間で VRRP プロトコル[6]により VIP を共有している。この構成では、ロードバランサ用 Pod とゲートウェイとの間で ARP 通信ができることが必要で、また他の Pod とマルチキャスト通信ができることも必要である。従って、Fig. 3 中の cbr0 と eth0 はブリッジ接続される必要があり Pod の IP アドレスはゲートウェイと同じブロードキャストドメインになければならない。

KVM 環境では、このような構成をとることで、負荷分散機能、フェイルオーバーとも問題なく動作した。

4.2 AWS EC2

Fig.4 に AWS EC2 上でのロードバランサの構成図を示す。外部からパブリックな固定 IP(AWS では EIP[7]と呼ばれる)に向けて送られた IP パケットは、ゲートウェイ上で任意のプライベート IP(VIP とする)宛のパケットに DNAT を用いて変換される。AWS EC2 環境で EIP 宛のパケットの DNAT 変換先は API を通して変更可能である。また VIP のアドレスアサインも同様に変更可能である。AWS EC2 環境では ARP が使えないが以上の仕組みを利用して、外部からの IP パケットを Pod へ到達させることが可能である。

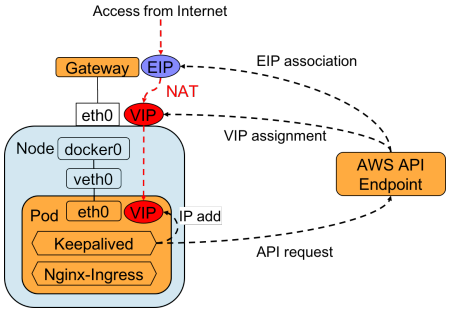

Fig.4 AWS EC2 環境ロードバランサネットワーク構成

ロードバランサの冗長化に関しては、AWS EC2 上ではマルチキャストを使うことができないので VRRP を用いることができない。VRRP にはユニキャストモードもあるが、あらかじめ相手の IP アドレスが決まっている必要があり、Kubernetes コンテナ環境では IP を指定して Pod 起動ができないので利用不可能である。従って、現状で冗長化は難しく、さらなる検討が必要である。

Kubernetes では Pod 起動時に動的に IP アドレスが付与されるが、AWS EC2 環境上では、API や WEB コンソールを介さずに付与された IP アドレスを用いると通信が制限されてしまう。そのため Pod 間の通信にはトンネリング型のオーバーレイネットワークが必要で、今回は flannel(udp,vxlan)[8]を利用した。

以上のような構成によりロードバランサの負荷分散機能の動作は確認できた。しかしながら、上述のようにフェイルオーバーについてはさらなる検討が必要であることがわかった。

5. 結論

本稿では、異なるクラウドプロバイダ間で Kubernetes システムをマイグレーションするためのロードバランサの実装方式を提案した。提案方式の有効性をオンプレミスの KVM 環境および AWS EC2 上で検証したところ、KVM 環境においては負荷分散機能、フェイルオーバーとも問題なく動作した。AWS EC2 においては、負荷分散機能は問題なかったが、フェイルオーバーについてはさらなる検討が必要であることが明らかになった。本問題を解決するために、今後、冗長化のメンバを知るためのメンバシップとメンバ間でのリーダーエレクションが可能な別のプロトコルについて検討する予定である。

6. 謝辞

本研究の一部は，JSPS 科研費 24240006 の助成を受けている。

参 考 文 献

[1] Marmol, Victor, Rohit Jnagal, and Tim Hockin. ″Networking in Containers and Container Clusters.″ *Proceedings of netdev 0.1, February* (2015).

[2] Linux-ip.net. (2016). *5.5.Destination NAT with netfilter (DNAT).* [online] Available at: http://linux-ip.net/html/nat-dnat.html [Accessed 15 Nov. 2016].

[3] Kivity, Avi, et al. ″kvm: the Linux virtual machine monitor.″ *Proceedings of the Linux symposium*. Vol. 1. 2007.

[4] GitHub. (2016). *nginxinc/kubernetes-ingress*. [online] Available at: https://github.com/nginxinc/kubernetes-ingress [Accessed 14 Nov. 2016].

[5] Keepalived.org. (2016). *Keepalived for Linux*. [online] Available at: http://www.keepalived.org/ [Accessed 14 Nov. 2016].

[6] RFC5798 | Hinden, R., Ed., ″Virtual Router Redundancy Protocol (VRRP)″, RFC 3768, DOI 10.17487/RFC3768, April 2004, <http://www.rfc-editor.org/info/rfc3768>.

[7] Docs.aws.amazon.com. (2016). *Elastic IP Addresses - Amazon Elastic Compute Cloud*. [online] Available at: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html [Accessed 15 Nov. 2016].

[8] GitHub. (2016). *coreos/flannel*. [online] Available at: https://github.com/coreos/flannel [Accessed 14 Nov. 2016].