コンテナ型仮想化基盤の構築およびサーバ仮想化のための利用

雨宮尚範

工学系技術支援室　情報通信技術系

概要

これまで、サーバ仮想化としては物理サーバ上に仮想マシンによるサーバを構築する方法がよく使われてきた。しかし、近年ではコンテナ型仮想化という方式でアプリケーションを動かすシステムが注目されている。2017 年には Open Container Initiative[1]によりコンテナの標準仕様が策定され、今後もさらに発展と普及が続くと見られている。

コンテナは仮想マシンを置き換えてしまうものではない。それぞれ使い分けることでより柔軟にシステムを構築することができる。本稿では、コンテナ型サーバの構築および運用についてある程度習得することを目指し、コンテナクラスタ環境の構築、仮想サーバの構築、管理ツールの機能確認を行った。

1　コンテナ型仮想化について

1.1　コンテナと仮想マシン

コンテナは OS 上の仮想的なユーザ空間を利用する技術である。これに対して、仮想マシンはハードウェアを含めたシステム全体をソフトウェアで再現する技術である。両者を比べると、コンテナには『オーバーヘッドが小さい』、『より効率的にリソースを使用できる』といった利点がある。仮想マシンにも OS 選択が自由であるなど別のメリットがあり、状況によってそれぞれを使い分けることが重要である。

コンテナは使い捨てたり負荷状況に応じて増減させるなど短いライフサイクルでの利用がよく行われている。そのような利用のためにはコンテナの構築や管理の自動化が重要であり、コンテナ管理ツールとして Kubernetes[2]が広く使用されている。

1.2　Kubernetes

Kubernetes はオープンソースソフトウェアのコンテナオーケストレーションツールでありコンテナの構築や管理を自動化することができる。Google によって開発された。競合ツールの開発終了やコンテナランタイムとして広く使われている Docker との統合[3]により、現在では Kubernetes が事実上の標準となっている。主要クラウドベンダーも Kubernetes マネージドサービスを提供している。

Kubernetes は単にコンテナそのものを取り扱うのではなく、管理のための概念的な要素が用意してある。そうした要素を利用に応じて定義することで自動的にコンテナの配備や更新が行われる。管理要素の例としては、管理上の基本単位であり複数のコンテナを内包しうる『Pod』、定義された数だけ Pod の複製をクラスタ上に維持する『ReplicaSet』、ReplicaSet を管理してローリングアップデート等を可能にする『Deployment』、Pod へのアクセス手段を提供する『Service』が挙げられる。

2　実験環境

2.1　ハードウェア構成

今回使用したサーバマシンの構成を表 1 に、ネットワークストレージの構成について表 2 に示す。

表 1. サーバマシンの構成

項目	説明
製品名	Shuttle DS61
CPU	Intel i3-2120T (2Core, 2.6Ghz)
メモリ	16GB
OS	CentOS 7

表 2. NAS の構成

項目	説明
製品名	QNAP TS-869 Pro
HDD	2TB x8
RAID	RAID6

2.2 ネットワーク構成

物理ネットワークの構成を図に示す。実験ではいくつかの仮想マシンを構築したが、仮想マシンのネットワークインターフェースはすべてブリッジ接続として設定した。

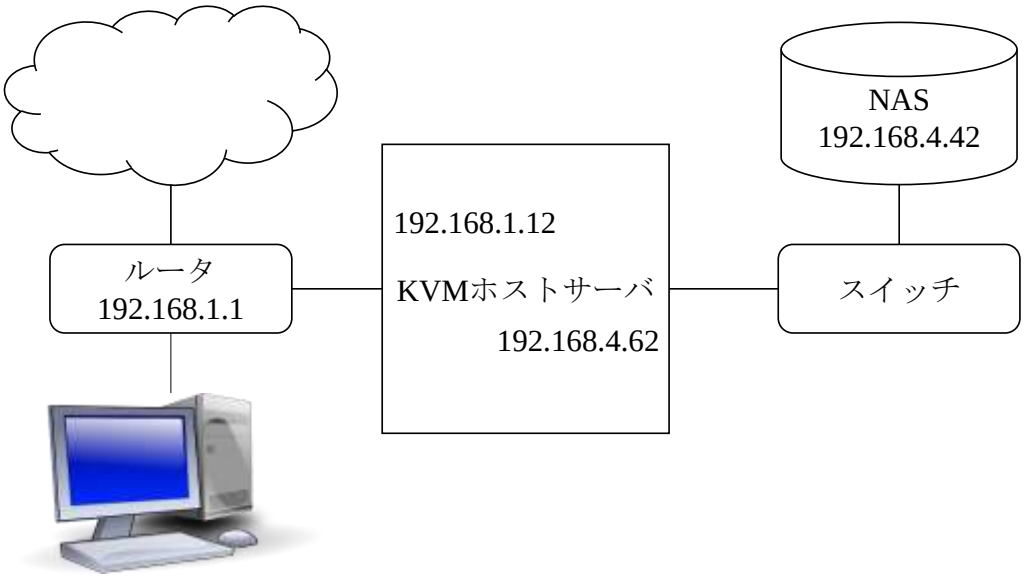

図 1. 実験環境の物理ネットワーク

2.3 Kubernetes 環境の構築

構築したコンテナクラスタを図 2 に示す。マスターサーバと各ノードおよび NAS はすべて KVM ホスト上に構築された仮想マシンである。構築手順は Kubernetes のドキュメント[4]を参考にしたが、インストールするパッケージについては RHEL のドキュメント[5]を参考にした。マスターサーバには Docker のプライベートリポジトリを構築し、各ノードには簡単のため非 TLS 接続で利用できるように設定を行った。

後述する kube-keepalived-vip[6]を利用するための準備として行った作業について述べる。特権コンテナの利用が必要になるため、マスターサーバおよびコンテナ配備予定のノード（node0）で特権コンテナの利用を許可する設定を行った。ServiceAccount の利用に関連して証明書の利用設定と認証方式の変更を行った（図 3）。プライベート認証局証明書とサーバ証明書は図 4 の手順で作成した。サーバ証明書には利用する可能性のあるすべてのホスト名および IP アドレスを subjectAltName で記載した。コンテナ配備を制御するため、各ノードには role ラベルを設定した。設定完了後マスターサーバおよび各ノードを再起動した。

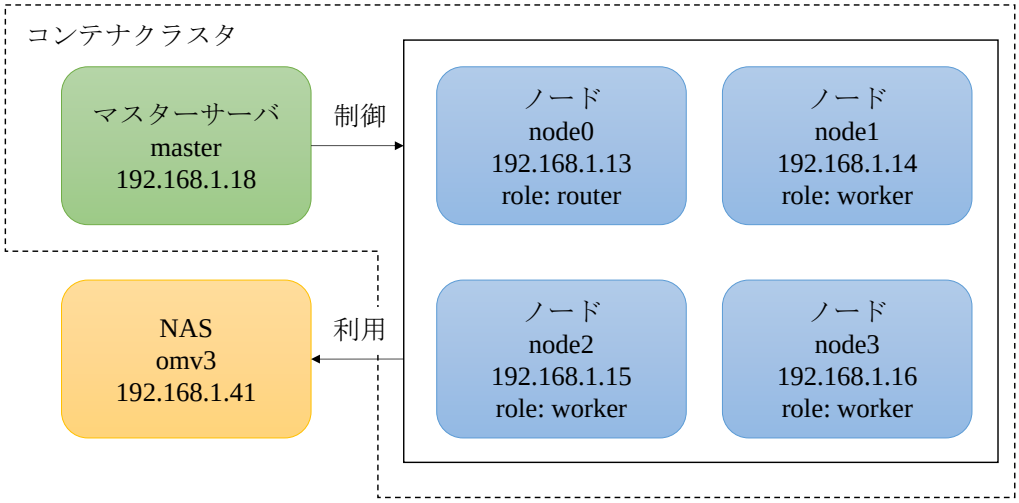

図 2. コンテナクラスタの構成

```
KUBE_API_ARGS="--authorization-mode=RBAC
  --tls-ca-file=/etc/kubernetes/certs/ca.crt
  --tls-cert-file=/etc/kubernetes/certs/apiserver.crt
  --tls-private-key-file=/etc/kubernetes/certs/apiserver.key"

-----

KUBE_CONTROLLER_MANAGER_ARGS="--root-ca-file=/etc/kubernetes/certs/ca.crt
  --service_account_private_key_file=/etc/kubernetes/certs/apiserver.key"
```

図 3. apiserver の設定（上）controller-manager の設定（下）

```
# cd /etc/kubernetes/certs
# cat <<EOF > openssl_ext.cnf
subjectAltName = @alt_names
[ alt_names ]
DNS.1 = apiserver.test
DNS.2 = apiserver
DNS.3 = master.test
DNS.4 = master
DNS.5 = localhost.localdomain
DNS.6 = localhost
IP.1  = 10.254.0.1
IP.2  = 192.168.1.18
IP.3  = 127.0.0.1
EOF
# openssl req -newkey rsa:2048 -nodes -sha256 -x509 -days 3650 \
    -keyout ./ca.key -out ./ca.crt -extensions v3_ca -subj "/CN=Kubernetes CA"
# openssl req -newkey rsa:2048 -nodes -sha256 \
    -keyout ./apiserver.key -subj "/CN=apiserver" | \
  openssl x509 -req -days 3650 -out ./apiserver.crt \
    -CA ./ca.crt -CAkey ./ca.key -CAserial /dev/null -set_serial $(date +%s) \
    -extfile ./openssl_ext.cnf
```

図 4. 証明書ファイルの作成

2.4 Dashboard[7]のインストール

Kubernetes では Web UI（Dashboard）を利用することができる。インストールした Kubernetes のバージョン（v1.5.2）に合わせて今回は Dashboard v1.5.1 を導入した（図 5）。使用した設定についてはダウンロードした公式の設定ファイルを編集し環境に合わせてコンテナの引数『`--apiserver-host`』を設定した。

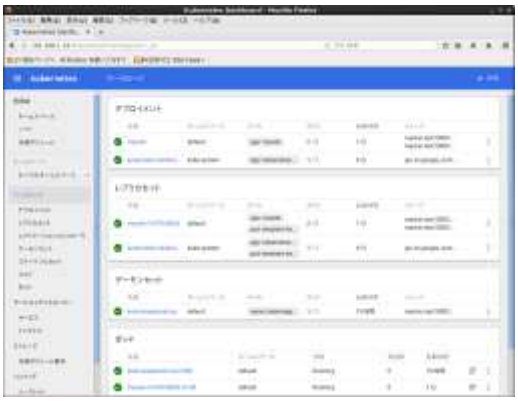

図 5. Dashboard

2.5 kube-keepalived-vip とその利用設定

構築したサーバをクラスタ外から利用できるようにするための設定は Kubernetes 環境をオンプレミスで構築した場合に突き当たる問題である。kube-keepalived-vip は Keepalived[8]と IPVS[9]（IP Virtual Server）でコンテナ上にロードバランサを構築することが可能である。Kubernetes と連携することで設定もある程度自動化されているため､比較的簡単に仮想サーバへのアクセスを実現できる｡Kubernetes では Ingress という HTTP(S)のロードバランサを利用することもできる。しかし、今回は SSH 接続できる仮想サーバの構築を想定していたため Ingress は検討から外れることになった。また、実際のところこの問題は Kubernetes を OpenStack[10]と連携させる、あるいは Enterprise Kubernetes とされる OpenShift[11]の利用に切り替えるといった方法で解決が可能である。しかし、今回は時間的余裕もなかったので Kubernetes コンテナクラスタで完結できる素朴な解決方法として kube-keepalived-vip を使用することにした。

kube-keepalived-vip の導入手順についてはドキュメントを参考にして行った｡公式で配布されているコンテナイメージではカーネルモジュールの読み込みがうまく行かず起動に失敗する問題があったが、CentOS 7 ベースのコンテナイメージを作成する（図 6）あるいはノード起動時に『ip_vs』モジュールを読み込むように設定することでうまく起動するようになった。今回使用した DaemonSet の設定については公式のものに以下の変更を加えた。コンテナイメージを CentOS 7 ベースのものに変更し、Dashboard に IP を割り当てるためコンテナの引数に『`--watch-all-namespaces`』を追加した。また配備されるノードを node0 に限定するため nodeSelector を role: router に設定した。

```
FROM docker.io/centos:centos7
RUN yum -y install keepalived iproute ipvsadm && \
    yum clean all                             && \
    rm /etc/keepalived/keepalived.conf
COPY kube-keepalived-vip /
COPY keepalived.tmpl /
COPY keepalived.conf /etc/keepalived
ENTRYPOINT ["/kube-keepalived-vip"]
```

図 6. CentOS ベースのイメージ構築用 Dockerfile

3 コンテナベース仮想サーバの構築

3.1 コンテナイメージの作成

今回想定した仮想サーバは Web サービスとコンテンツ編集用に SSH サービスを提供するものである｡コンテナイメージの構築にはいくつか考え方があるが、本稿では 1 つのサービスにつき 1 つのコンテナを割り当

てるというやり方をとった。イメージの作成に使用した設定を図 7 に示す。作成したイメージはマスターサーバのリポジトリに保存した。

```
FROM docker.io/centos:centos7
RUN yum -y install httpd && systemctl enable httpd
RUN sed -i -e '151s/None/Options/' -e '/^ *DirectoryIndex/s/$/ index.shtml/' \
  /etc/httpd/conf/httpd.conf
CMD ["/sbin/init"]
EXPOSE 80

---

FROM docker.io/centos:centos7
RUN yum -y install openssh-server && systemctl enable sshd
RUN useradd -u 1001 -m test; \
  install -d -o test -g test -m 700 /home/test/.ssh; \
  install -o test -g test -m 600 /dev/null /home/test/.ssh/authorized_keys; \
  echo "(SSH-PUBKEY)" >> /home/test/.ssh/authorized_keys
CMD ["/sbin/init"]
EXPOSE 80
```

図 7. Web サービス用 Dockerfile（上）、SSH サービス用 Dockerfile（下）

3.2 PersistentVolume（永続ボリューム）の利用設定

コンテナのライフサイクルは短いためコンテナ内部にデータを置いておくとすぐコンテナと一緒に消えてしまう。Kubernetes ではそうしたデータを保存するための領域を PersistentVolume として管理している。PersistentVolume は PersistentVolumeClaim（永続ボリューム要求）とその条件に応じて自動的に紐つけられ、Pod の設定で PersistentVolumeClaim の利用を定義することでコンテナにマウントされる。

今回は NAS 上に Web コンテンツ用のディレクトリを用意した。仮想サーバで利用する PersistentVolume として vol1 を作成し NFS で Web コンテンツ用のディレクトリにアクセスするように設定した。また、ポッドに割り当てる PersistentVolumeClaim として docroot を作成した。

3.3 コンテナの構築および配備

Deployment を定義して仮想サーバ用 Pod の構築と配備を行った。使用した設定ファイルを図 8 に示す。レプリカ（複製）の数は 2 で、nodeSelector で配備先のノードを限定する（role: worker）ようにした。設定通り 2 セットの Pod が node0 を除くいずれかのノード上に構築された。

3.4 外部アクセス用の設定および確認

kube-keepalived-vip の ConfigMap に新しく仮想サーバ公開用の設定を追加した（図 9）。その後、構築した Pod にアクセスするための Service を作成した（図 10）。使用した設定ファイルを図に示す。割り当てた IP（192.168.1.31）にアクセスするためには externalIPs で ConfigMap に記述したものと同じ IP を設定してやる必要があった。

kube-keepalived-vip の Pod に設定ファイルが生成されていることを確認した。Web コンテンツとしてホスト名を表示するページを追加し、仮想サーバ用 IP に対してブラウザでアクセスして Web ページが表示されることを確認した。また、リロードを繰り返すことで表示が切り替わり負荷分散されていることが確認できた（図 11）。

コンテナ構築時に設定したユーザ名および秘密鍵を使用して SSH で仮想サーバ用 IP にアクセスしたとこ

ろログインすることができなかった。調査したところノード（node0）の SSH デーモンにより 22/TCP ポートが IP アドレス ANY で LISTEN（接続待ち）状態になっていたのが原因であった。ノード側の SSH デーモンを停止させたところコンテナにログインすることができた。また、Web コンテンツを編集して表示に反映されることを確認した。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: myweb
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: myweb
    spec:
      volumes:
        - name: docroot
          persistentVolumeClaim:
            claimName: docroot
      containers:
        - name: httpd
          image: master.test:5000/myrepo/httpd:v1
          ports:
            - containerPort: 80
          volumeMounts:
            - mountPath: "/var/www/html"
              name: docroot
        - name: sshd
          image: master.test:5000/myrepo/sshd:v1
          ports:
            - containerPort: 22
          volumeMounts:
            - mountPath: "/var/www/html"
              name: docroot
      nodeSelector:
        type: worker
```

図 8. 仮想サーバ用 Deployment 設定ファイル

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vip-configmap
data:
  192.168.1.31: default/myweb
  192.168.1.33: kube-system/kubernetes-dashboard
```

図 9. 仮想サーバ公開用 ConfigMap 設定ファイル

```
apiVersion: v1
kind: Service
metadata:
  name: myweb
spec:
  externalIPs:
    - 192.168.1.31
  ports:
    - name: httpd
      port: 80
      protocol: TCP
    - name: sshd
      port: 22
      protocol: TCP
  selector:
    app: myweb
```

図 10. 仮想サーバ用 Service 設定ファイル

図 11. 負荷分散された Web サービス

4　Kubernetes の機能について確認

Kubernetes の提供する機能のうちフェイルオーバーとローリングアップデートについて動作を確認した。今回構築した仮想サーバでは Deployment 配下の ReplicaSet の機能でフェイルオーバー、Deployment の機能でローリングアップデートを利用できる。

4.1　フェイルオーバー

フェイルオーバーは稼働中のシステムで障害が発生したとき別のシステムが自動的に機能を引き継ぐことである。Kubernetes では Pod のフェイルオーバーが可能である。以下で実際の作業と結果について述べる。まず仮想サーバの Pod が動作しているノードを停止させた。40 秒ほどでノードの状態が Unknown に変化した。その後 5 分ほどして Pod の状態が Unknown に変化して別のノードに新しく Pod が作成された。状態が Unknown に変化した Pod はノードの復旧後に消えた。障害発生から回復までブラウザでページ更新を繰り返したが Web ページは問題なく表示され続けた。

4.2　ローリングアップデート

ローリングアップデートはシステム全体を停止させることなく更新を行うことである。Kubernetes ではレプリカをもつ Pod について一部ずつ交換していくという形になる。以下で実際に行った作業と結果について述べる。まず準備としてコンテナイメージのタグを変更して新しくリポジトリに保存した。アップデートの実行は管理コマンドで Deployment に設定したコンテナイメージのタグを新しいものに変更して行った。変更後すぐに新しいイメージに対応した ReplicaSet が生成された。Pod の更新は、新しいイメージの Pod が 1 つ生成され起動完了すると古いイメージの Pod が 1 つ廃棄されるという形で 1 つずつ置き換わっていた。動作中の Deployment の設定で spec.strategy.rollingUpdate を調べ、割り当てられた設定通りの挙動であるということを確認した。

5　今後の課題

これまでに仮想サーバの構築までを確かめたが、まだ運用に際してはいくつか課題が残っている。例えばログの採取や保存方法、バックアップのしかた、マスターサーバの冗長化である。フェイルオーバーやロー

リングアップデート以外についても詳しく挙動を確認しておきたい。また、今回は見送ったが OpenStack との連携や OpenShift への切り替えも検討する必要がある。

6　まとめ

本稿では、仮想マシン 5 台からなるコンテナクラスタ環境を構築した。コンテナによる仮想サーバの構築方法を検討するとともに管理ツールである Kubernetes のしくみの一部について挙動を確かめた。実際に仮想サーバとして運用するにはまだ検討すべき事項が残っているが、コンテナ型仮想サーバに触れて仮想マシンとの扱いや考え方の違いについて学べたことは有意義であった。今回学んだことを活かして実際の業務にも役立てていきたい。

7　謝辞

本稿執筆の機会を用意していただいた技術研修会実行に携わられた方々、進捗の確認をしていただいた藤原課長に感謝する。また、今回の研鑽にあたり工学部情報工学コースの機器を利用させていただいた。

参考文献

[1] Open Container Initiative, https://www.opencontainers.org/ (2018 年 2 月 14 日閲覧)

[2] Kubernetes, https://kubernetes.io/ (2018 年 2 月 14 日閲覧)

[3] Docker Platform and Moby Project add Kubernetes, https://blog.docker.com/2017/10/kubernetes-docker-platform-and-moby-project/ (2018 年 2 月 14 日閲覧)

[4] CentOS, Kubernetes, https://v1-5.docs.kubernetes.io/docs/getting-started-guides/centos/centos_manual_config/ (2018 年 2 月 14 日閲覧)

[5] コンテナーの使用ガイド, Red Hat Customer Portal, https://access.redhat.com/documentation/ja/red-hat-enterprise-linux-atomic-host/version-7/getting-started-with-containers/#creating_a_kubernetes_cluster_to_run_docker_formatted_container_images (2018 年 2 月 14 日閲覧)

[6] Dashboard, https://github.com/kubernetes/dashboard/ (2018 年 2 月 14 日閲覧)

[7] Keepalived, http://www.keepalived.org/ (2018 年 2 月 14 日閲覧)

[8] IPVS, http://www.linuxvirtualserver.org/software/ipvs.html (2018 年 2 月 14 日閲覧)

[9] kube-keepalived-vip, https://github.com/kubernetes/contrib/tree/master/keepalived-vip (2018 年 2 月 14 日閲覧)

[10] OpenStack, https://www.openstack.org, (2018 年 2 月 14 日閲覧)

[11] OpenShift, https://www.openshift.com, (2018 年 2 月 14 日閲覧)