コンテナ型仮想化アプリケーションの展開・管理自動化ツール

Kubernetes の学習

○島田啓史

情報通信技術支援室　情報システム構築技術グループ

概要

工学研究科・工学部技術部情報通信技術系では多くの仮想サーバを管理しており、今後もさらなる集約化により仮想サーバが増え、管理が煩雑になる可能性がある。このような傾向に対応する手段として近年、コンテナ型仮想化を用いて、効率よくサーバ管理・構築を行う仕組みが注目を集めている。そこで今回、コンテナのデプロイ（展開）、管理システムである Kubernetes について学習し、実際に複数個のコンテナの展開について、検証を行ったので報告する。

1　コンテナ型仮想化について

現在広く使用されているサーバ仮想化と、コンテナ型仮想化との比較を図 1 に示す。コンテナ仮想化ソフトウェアにより、アプリケーションが利用する領域がコンテナ単位に分割される。サーバ仮想化に比べて、ゲスト OS が存在しないため、アプリケーション実行時に必要なメモリ容量・CPU 負荷が軽減され、システム資源の使用効率が良くなる。

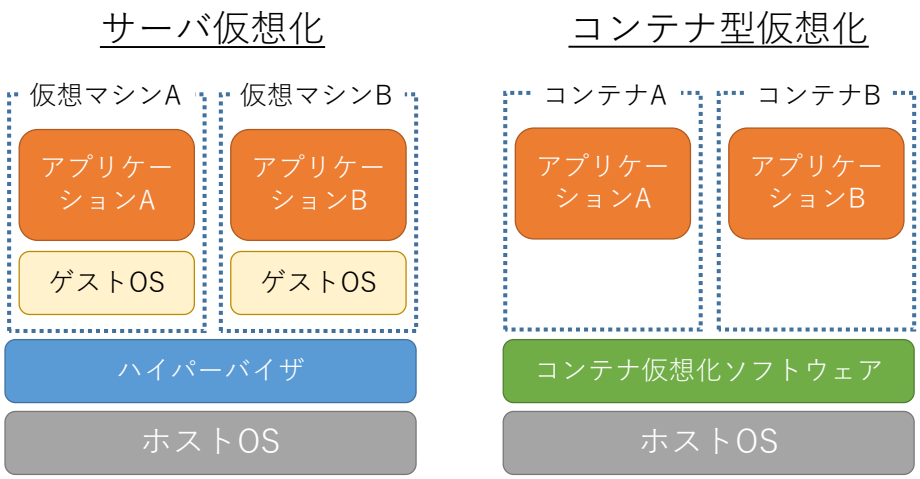

図 1. サーバ仮想化とコンテナ型仮想化

最もよく利用されているコンテナ仮想化ソフトウェアとしては Docker が挙げられる。しかし、Docker だけではコンテナの大規模な展開に対応していないことから、効率的に管理・デプロイするためのプラットフォームとして近年利用されているのが Kubernetes である。Kubernetes はホスト OS 上で動作し、内部で Docker を呼び出す。また、複数のサーバで稼働する Kubernetes を連携させて、同じコンテナを分散実行することも可能である。

2　Kubernetes について

2.1　マニフェスト

Kubernetes では、YAML 言語で記述されたマニフェストを用いてコンテナのデプロイを行う。マニフェストの種類として、1) Deployment（デプロイ設定：コンテナの定義）、2) Service（ポート番号情報の設定）、3) ConfigMap（環境変数、及び設定ファイルの定義）、4) PersistentVolumeClaim（ストレージ領域の確保）等があり、複数個のマニフェストを組み合わせてコンテナをデプロイする。

一例として、Web サーバコンテナ(nginx)をデプロイするための Deployment マニフェストを図 2 に示す。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: example1
  name: example1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: example1
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: example1
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - mountPath: /usr/share/nginx/html/
          name: hello-world-html
      volumes:
      - name: hello-world-html
        configMap:
          name: hello-world-configmap
          items:
          - key: index.html
            path: index.html
```

起動するコンテナ名を指定

下記の volumes: でマッピングした index.htmlを
コンテナ内の/usr/share/nginx/html/ に置く

ConfigMapマニフェストで定義した内容を、
index.html ファイルにマッピングする

図 2. Deployment マニフェストの例

2.2　Pod

Kubernetes では、1 つ以上のコンテナの集合を Pod という単位で表し、1 つのデプロイは 1 つ以上の Pod で構成される。Pod 内では IP アドレスが共有され、ポート番号を重複しないように設定することで、コンテナ同士で通信することが可能である。負荷分散のために、同じ構成の Pod を複数稼働させたい場合は、Pod のレプリカ数を定義することができる。デプロイ、Pod、コンテナの関係を図 3 に示す。

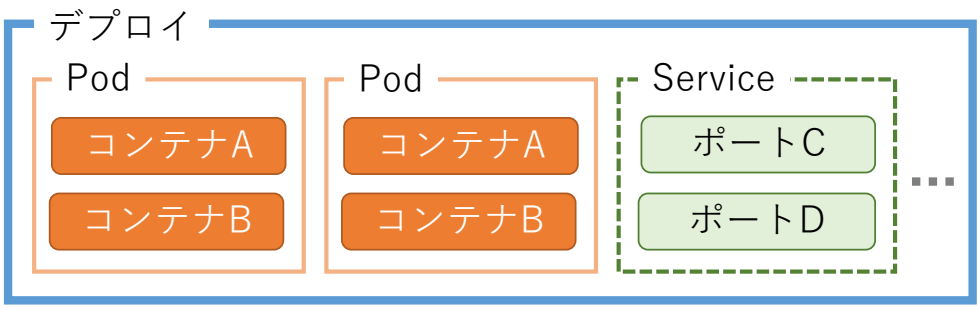

図 3. デプロイ、Pod、コンテナの関係

Pod のデータは、ホスト OS から分離された仮想ファイルシステムに格納される。デプロイの終了時に仮想ファイルシステムは削除されるため、データの永続化が必要な場合は、あらかじめ永続的なストレージ領域を確保し、コンテナにマウントするためのマニフェストを作成する必要がある。

3　Kubernetes の学習

3.1　Minikube 環境の構築

今回の検証では、練習・開発検証用 Kubernetes 実装である Minikube を使用した。また、物理サーバを複数台用意できなかったため、サーバ仮想化を用いて検証環境を構築することにした。具体的には、一台の物理サーバに VMware ESXi ハイパーバイザをインストールし、Minikube 用・NFS 用・Docker プライベートリポジトリ用の 3 つの仮想サーバを作成した。コンテナイメージは通常、インターネット上の Docker Hub からダウンロードされるが、Docker Hub にアップロードすることができないカスタムコンテナを使用することを想定し、Docker プライベートリポジトリサーバを構築した。NFS サーバは、コンテナが利用する永続的なストレージ領域を提供するために構築した。Minikube 用仮想サーバには、Docker, kubectl, Minikube をインストールし、ファイアウォール設定を行った後、Minikube を起動した。

図 4 に検証環境のシステム構成図を示す。仮想サーバのゲスト OS には、いずれも CentOS 7 を使用した。

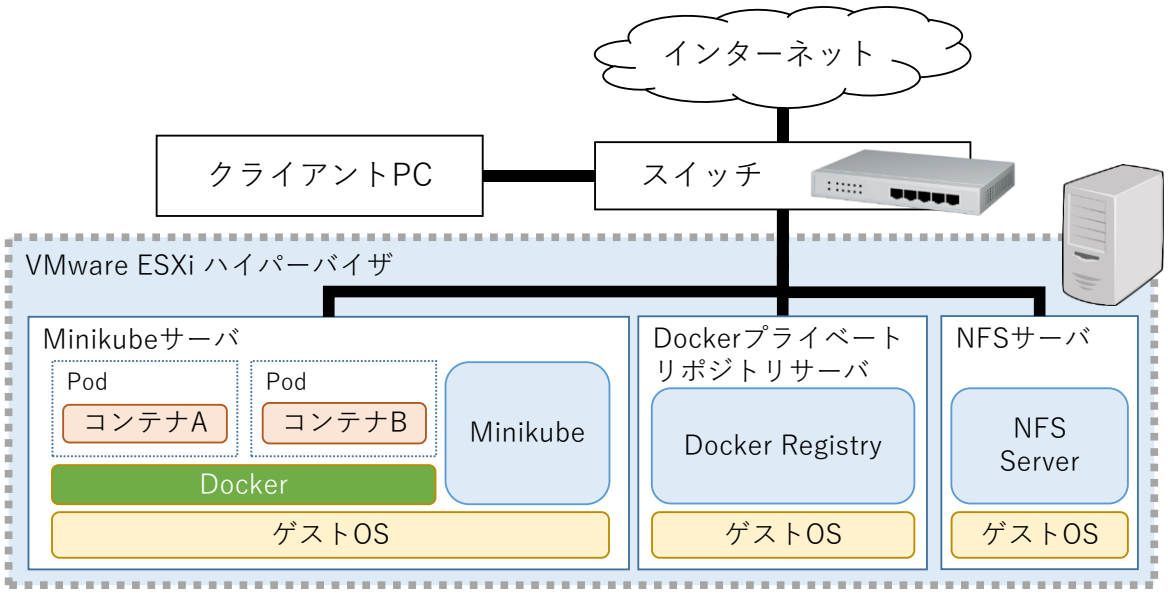

図 4. 検証環境のシステム構成

3.2　Web サーバコンテナ(nginx)のデプロイ（検証 1）

マニフェストの記述方法と Minikube のコマンドを理解するために、まず Web サーバコンテナ(nginx)を 1 つだけデプロイした。

Web サーバ自体の設定ファイルや表示する HTML ファイルについては、ConfigMap マニフェストファイル(example1-hello.yaml)に定義を行い、別途用意した Deployment マニフェストファイル(example1.yaml)から参照させることで、コンテナの稼働時にあらかじめ配置することにした。同様に、開放するポート番号については、Minikube 用仮想サーバの TCP 30812 番ポートにアクセスがあった場合、Pod に割り当てられたプライベート IP アドレスの TCP 80 番ポートにアクセスさせるよう、Service マニフェストファイル(example1-service.yaml)に定義した。

次に、作成したマニフェストファイルを指定して kubectl コマンドを実行し、Web サーバコンテナ(nginx)が正常に稼働したことを確認した（図 5)。その後、クライアント PC の Web ブラウザを用いて、コンテナに正常にアクセスできることが確認できた。

```
# # デプロイを実行
# kubectl create -f example1.yaml -f example1-hello.yaml -f example1-service.yaml
deployment.apps/example1 created
configmap/hello-world- created
service/example1-service created
# # デプロイ状況の確認
# kubectl get deployments
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
example1                  1/1     1            1           33s
# # Podの確認
# kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
example1-6db8d95948-fpqld             1/1     Running   0          40s
# # サービスの確認
# kubectl get services
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
example1-service     NodePort    10.108.155.160   <none>        80:30812/TCP     50s
```

図 5. デプロイの実行とコンテナの稼働確認

3.3　スケーリングと複数コンテナのデプロイ（検証 2）

次に、Kubernetes が持つスケーリング機能の検証を行った。具体的には、検証 1 で作成したマニフェストファイルを基に、Pod のレプリカ数を 10 として、負荷分散の状態を検証することにした。この時、Web サーバコンテナ(nginx)のみでは負荷分散の状態を確認することが難しいため、PHP スクリプトを用いて、各 Pod の内部 IP アドレスを表示することにした。検証構成を図 6 に示す。

1 つの Pod 内に、Web サーバコンテナ(nginx)と PHP スクリプトエンジンコンテナ(php-fpm) を作成するように Deployment マニフェストファイルを変更し、2 つのコンテナが連携する設定を記述する ConfigMap マニフェストファイルを定義した。

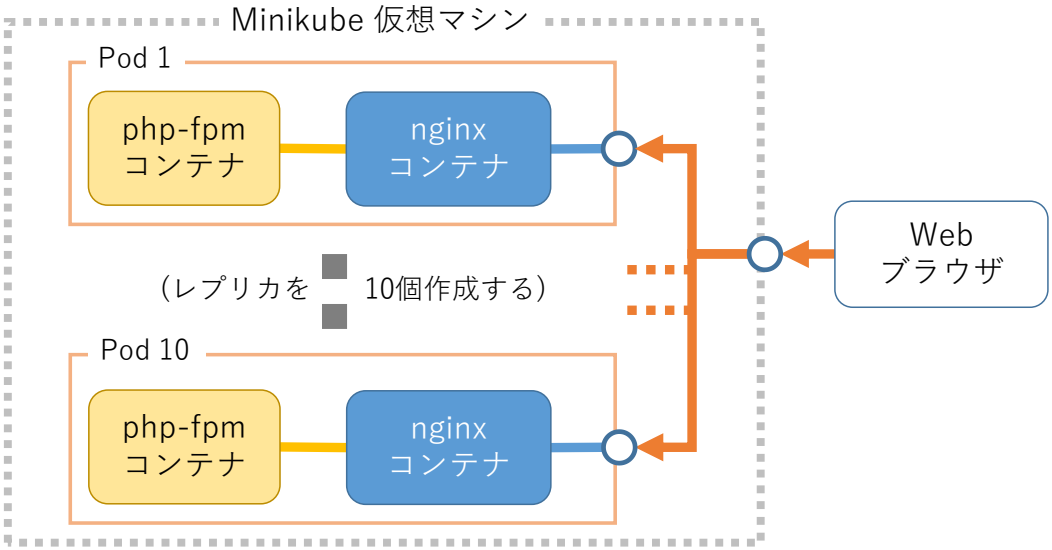

図 6. スケーリング機能の検証構成

検証 1 と同様に、作成したマニフェストファイルを指定して kubectl コマンドを実行し、コンテナをデプロイした後、Web ブラウザからアクセスを行った。結果として、リロード毎に表示される IP アドレスが変化し、ブラウザからのアクセスが 10 個の Pod に分散されている事が確認できた。

3.4　永続的なストレージを使用したコンテナのデプロイ（検証 3）

検証 2 で作成したマニフェストファイルを基に、実際のホスティングサービスを想定し、NFS サーバのス

トレージ領域を使用した Web サーバコンテナ(nginx)を構築した。

コンテナをデプロイする前に、Minikube から NFS サーバに接続するための設定を行った。Kubernetes 標準の NFS クライアントドライバは、デプロイ毎に領域を分割するダイナミック・プロビジョニング機能に対応していないため、NFS Client Provisioner アドオンを Minikube 用仮想サーバに別途インストールした。次に、表 1 に挙げる 4 つのマニフェストファイルを作成し、コンテナのデプロイを行った。このとき、Web 管理者が Web コンテンツをアップロードするための接続経路として、SFTP サーバコンテナ(atmoz/sftp)も同時にデプロイした。検証構成を図 7 に示す。

表 1. 用意したマニフェスト

マニフェストの種類	設定の内容
PersistentVolumeClaim	NFS サーバからストレージ領域を確保する
ConfigMap	Web サーバコンテナ(nginx)と PHP スクリプトエンジンコンテナ(php-fpm)を接続する
Service	Minikube 用仮想サーバの TCP 30001 番ポートにアクセスすると Pod 内の TCP 80 番ポートに、TCP 30022 番ポートにアクセスすると Pod 内の 22 番ポートにアクセスする
Deployment	他のマニフェストの内容をコンテナ内に設定し、1)Web サーバコンテナ(nginx)、2) PHP スクリプトエンジンコンテナ(php-fpm)、3) SFTP サーバコンテナ(atmoz/sftp) の 3 つを稼働

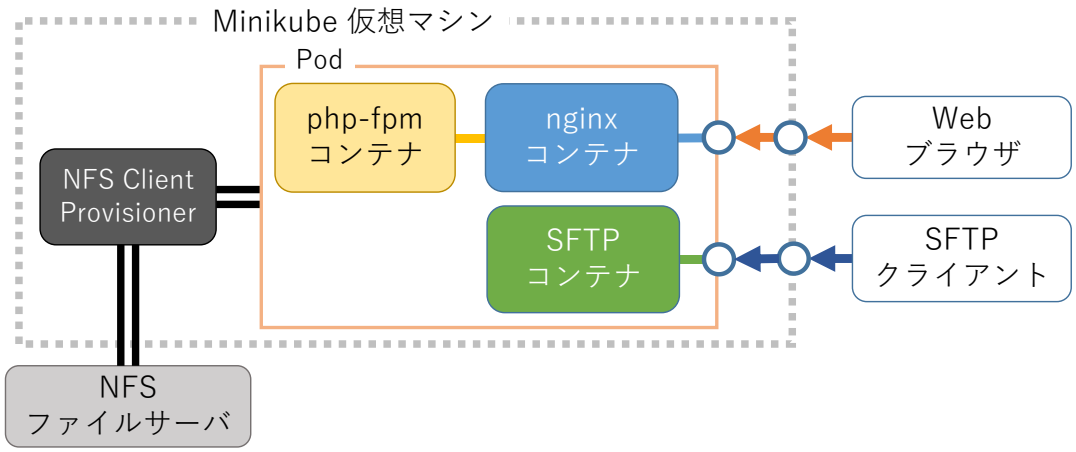

図 7. 永続的なストレージと検証構成

コンテナをデプロイした後、SFTP クライアントで SFTP サーバコンテナ(atmoz/sftp)に接続してファイルのアップロードを行い、NFS サーバ上にファイルができていることを確認した。次に Web ブラウザでアクセスし、アップロードしたファイルを閲覧できることが確認できた。

3.5 カスタムコンテナのデプロイ（検証 4）

最後に、実際の業務を想定し、カスタムコンテナをデプロイする検証を行った。本検証で作成、使用したカスタムコンテナは、Web サーバコンテナ(nginx)イメージ内の設定ファイル server.conf を変更しただけの簡単なものとした。server.conf を用意し、カスタムコンテナを作成し、Docker プライベートリポジトリサーバにアップロードした（図 8）。

```
$ # 以下の内容をDockerfileに入力
$ cat <<EOF > Dockerfile
FROM nginx:latest
MAINTAINER etech-shimada
ADD server.conf /etc/nginx/conf.d/server.conf
EOF
$ # Dockerfileからコンテナを作成
$ sudo docker build -t etech-shimada/nginx .
$ # Dockerプライベートリポジトリサーバにアップロード(IPアドレスは例)
$ sudo docker tag etech-shimada/nginx:latest
    192.168.100.222:31333/etech-shimada/nginx:latest
```

図 8. カスタムコンテナの作成とアップロード手順

Docker プライベートリポジトリサーバ内のカスタムコンテナを指定するために、Deployment マニフェストファイル内で、カスタムコンテナ名の前に IP アドレス情報を追加した。次に Docker 側の設定ファイルで、Docker プライベートリポジトリサーバからの非 SSL アクセスを許可し、デプロイした。その後、クライアント PC の Web ブラウザを用いて、カスタムコンテナに正常にアクセスできることが確認できた。

4　おわりに

Kubernetes の基礎的な使用方法を学習し、具体的にいくつかの場面を想定したマニフェストを作成、デプロイすることで、近年のシステム開発のトレンドである、インフラのコード化を体感することができた。

今後の課題として、以下の事項が挙げられる。

- マニフェスト作成時、記述誤りを即時に検出するツールの調査
- 実環境により近い、他の Kubernetes 実装の検証
- 高度なマニフェストの検証（パスワード格納、ロードバランサ等）

参考文献

[1] 石澤基，五十嵐綾，大塚元央，須田一輝，稲津和磨，九岡佑介，坂部広大，青山真也，池添明宏，上岡真也 著『みんなの Docker/Kubernetes』技術評論社, 2019 年

[2] 須田一輝，稲津和磨，五十嵐綾，坂下幸徳，吉田拓弘，河宜成，久住貴史，村田俊哉　著『Kubernetes 実践入門 プロダクションレディなコンテナ&アプリケーションの作り方』 技術評論社, 2019 年

[3] 青山真也 著『Kubernetes 完全ガイド』インプレス, 2018 年

[4] 中井悦司 著『Docker 実践入門―Linux コンテナ技術の基礎から応用まで』技術評論社, 2015 年