Kubernetes

盛 滉佑
Kosuke MORI

1　はじめに

近年，Docker の登場により，アプリケーションにおけるインフラ構築が簡素化した．Docker とは，コンテナ型の仮想化環境を提供する提供するオープンソースソフトウェアである．コンテナ型仮想化とは，複数の OS を立ち上げることなくアプリケーションを実行する仕組みである．（以下，コンテナ型仮想化により生成される実行環境をコンテナと呼ぶ.）コンテナ型仮想化の普及により，アプリケーションの開発・運用における管理コストが減少した．

Docker の登場と同時期に，コンテナ型仮想化を用いたアプリケーション構築において，マイクロサービスという開発手法が普及した．マイクロサービスとは，複数の独立した機能を組み合わせることで，一つの処理を実現するアーキテクチャ（システム構成）である．マイクロサービスが普及した要因として，クラウドの出現とサービスの多様化が挙げられる．近年のアプリケーション開発におけるクラウド活用の動向に伴い，モジュールが独立して展開可能，かつ増築可能（スケーラブル）なアプリケーションが必要とされている．マイクロサービスをアプリケーション構築に用いることで，クラウド上で快適に運用可能なアプリケーションの構築が可能となる．しかし，マイクロサービスがアプリケーション構築の主流となるにつれ，単位アプリケーションあたりが扱うコンテナの数は膨大になり，コンテナ間の関係が複雑化した．

そこで，煩雑化したコンテナを一元管理するツールが，Kubernetes である．本稿では，Kubernetes の概要と利点に関して記述する．

2　Kubernetes

2.1　概要

Kubernetes とは，オープンソースのコンテナオーケストレーションシステムである．コンテナオーケストレーションシステムとは，大量のコンテナを自律的に管理するソフトウェアである．Kubernetes の開発元は Google であり，2014 年に公開された．現在は Cloud Native Computing Foundation（CNCF）が保守・管理を行っている．Docker を含む多数のコンテナツールと連携して動作する．コンテナオーケストレーションシステムには，Amazon ECS や Docker の Swarm などがあるが，現在 Kubernetes がデファクトスタンダード（事実上の標準）と呼ばれる存在と見なされている．また，Kubernetes は，マイクロサービスをベースとしたアプリケーションの実装方法として認知されている．

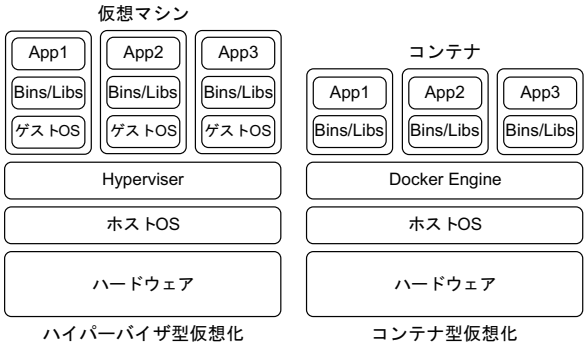

Fig.1　仮想マシン簡易図

2.2　コンテナ型仮想化の課題

仮想化技術として，ハイパーバイザ型仮想化とコンテナ型仮想化がある．ハイパーバイザ型仮想化とコンテナ型仮想化の簡易図を Fig. 1 に示す．コンテナ型仮想化は従来の仮想化と比較し，ディスク使用量が少なく，環境構築や起動が速いという利点がある．しかし，コンテナ型仮想化の普及につれ，様々な課題が出現した．コンテナ型仮想化を運用するにあたって発生した主な課題を四つ挙げる．

- 自ホスト上のコンテナの管理しかできない
- ホストに障害が発生した場合，冗長性の確保が難しい
- ソフトウェアを連携させる場合，各コンテナのネットワーク情報や，連携情報の管理に手間がかかる
- 大量のコンテナを全て監視しなければいけない

以上の課題を解決するために，コンテナオーケストレーションシステムである Kubernetes が必要となる．

2.3　Kubernetes の利点

Kubernetes の基本的な機能は，コンテナの運用・管理・監視からなる．主な利点として，下記の四つが挙げられる．

- コンテナの自動スケジューリング
  CPU やメモリの状態を見て，自動的に適切なホストを選択し，コンテナを起動する．起動するコンテナの数も容易に指定できる．自動スケジューリングによって，コンテナ管理にかかるコストを削減できる．
- ソフトウェアアップデートにおけるリスクを低減
  コンテナを複製することで容易に検証環境を作成できる．よって，本番と全く同じ環境であらかじめ動作確認を行うことが可能である．また，コンテナを複製することでシステムを停止せずアップデートを実行できる．以上の機能より，ソフトウェアアップデート時に起こる動作不良などのリスクを低減できる．

- 自己修復による高い耐障害性
  コンテナの状態を常に監視する．よって，万が一コンテナのプロセスが停止した場合，再度コンテナを起動し，自己修復する．ホストに問題が起こり，ノード障害が発生した場合でもサービスを継続できる．
- ネットワークの仮想化によりホスト間通信を実現
  コンテナ型仮想化は自ホスト以外のコンテナの管理が容易でない．Kubernetes を活用することで，複数のノードで構成される環境を一つの実行環境のように扱うことができる．よって，別のホストにあるコンテナもまとめて管理できる．

以上の主な特徴により，コンテナ型仮想化の課題を克服し，さらに安定的にサービスを提供することが可能となる．

2.4 Kubernetes の構成

Kubernetes において，同じ記憶領域を共有する一つ以上のコンテナの集まりを Pod と定義する．Kubernetes は，管理上の最小単位として Pod を扱っている．Pod としてコンテナをグループ化することで，効率的にコンテナを管理できる．また，Kubernetes では，Pod それぞれに識別のためユニークな Pod IP アドレスが振られる．したがって，自律的に IP アドレスが割り振られることで，アプリケーションはポートの衝突の危険を考える必要がない．

また Kubernetes は，マスタ・スレーブアーキテクチャで構成される． マスタ・スレーブアーキテクチャとは，マスタとなる一つのプロセスが 他の一つまたは複数のプロセス（スレーブ）を一方的に制御するシステム構造である．具体的には，コンテナを実際に運用するワーカノードと，各ワーカノードを監視・管理するマスタノードに分けられる ．クライアントはマスタノードを介して各ワーカノード上のプロセスにアクセスできる．Kubernetes のアーキテクチャ簡易図を Fig. 2 に示す．

マスタノードは，システムの稼働状況とシステム全体に渡る直接的な通信を管理する．マスタノードの構成は主に下記の要素から成り立つ．

- API server : Kubernetes API を提供，Kubernetes に対するすべての操作は，API server を経由する
- etcd : 各種設定情報を格納するデータストア
- Scheduler : スケジューリングプロセス（Pod の監視）
- Controller Manager : Pod 群を管理するコントローラを実行するプロセス

ワーカノードでは，Pod が動作している．マスタノードが Pod の管理を行い，Pod 内のコンテナはワーカノード上にデプロイ（実行可能な実行）される．ワーカノードの構成は主に下記の要素から成り立つ．

- Kubelet : 各ノードの実行状態に対する責務を担う．指示を受け，コンテナ群を Pod として組織，管理する
- Kube-Proxy : ホスト上のネットワークルールを管理し，適切なコンテナにアクセスを中継する
- Pod : コンテナ化したアプリケーションを格納

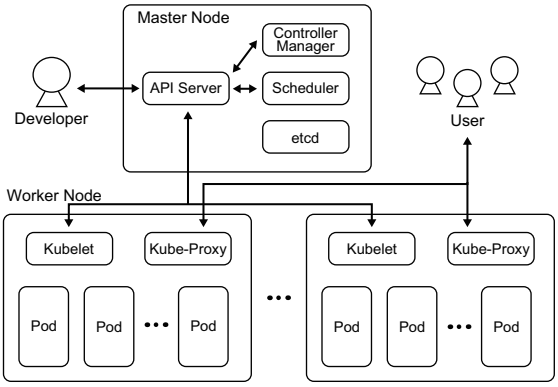

Fig.2　アーキテクチャ簡易図

以上より，マスタ・スレーブアーキテクチャを採用することで，Kubernetes による自律的機能が実現する．

3 Kubernetes の導入事例

Niantic, Inc. は，Pokemon Go のインフラ構築に Kubernetes 採用した．2016 年リリース当初，Pokemon Go へのアクセス数は予想の 50 倍に相当した．しかし Kubernetes により，稼働に要する各サービスを即座にスケールアウトし，サービス停止に至ることなく，システム障害を最小限に抑えた．

また，Mercari, Inc. は，自社サービスのマイクロサービス化のため Kubernetes を導入した．従来，Mercari, Inc. は，自社サービスを一元管理していた．しかし，Kubernetes による拡張可能な基盤システムに置き換えることで，開発・運用体制に柔軟性を持たせることが可能となった．その結果，拡張性の担保だけでなく，開発における生産性の向上や開発人材の確保という面からも効果があった．

4 今後の展望

2017 年に Docker が Kubernetes の統合を発表した．この発表に伴い，AWS，Azure，Google Cloud，IBM Cloud などの主要クラウドベンダは Kubernetes のマネージドサービスを開始した．以上の背景から，今後 Kubernetes が大規模サービスで活用されることが期待できる．

また，今後 Kubernetes のレイヤで束ねてコンテナをクラウド間で柔軟に移動できるようになる可能性がある．この技術が実現すれば， ユーザは Kubernetes を用いて，稼働状況や利用価格に合わせてクラウドのリソースを自由に使い分けることが可能となる．柔軟なリソースの選択が実現すれば，Kubernetes 導入コストが減少し，今まで以上にインフラ構築にかかるコストは減少すると予想される．

参考文献

1) Bringing Pokémon GO to life on Google Cloud - Google, https://cloud.google.com/blog/products/gcp/bringing-pokemon-go-to-life-on-google-cloud, 参照 Apr.23, 2019
2) メルカリの導入事例 - Google Cloud Platform Japan, https://cloudplatform-jp.googleblog.com/2018/01/Google-Cloud-Platform-Mercari-kubernetes.html, 参照 Apr.25, 2019