@IT
atmarkIT

これから始める企業のためのコンテナ実践講座

矢野哲朗 棚田美寿希, スタイルズ [著]

「コンテナって何？ どう使える？」——ソフトウェア開発の課題を解決するコンテナ技術

「コンテナ技術」やコンテナ実行環境の「Docker」、大量のコンテナ管理や負荷分散を実現する「Kubernetes」について概要から本番活用の仕方まで解説する本連載。第 1 回は「コンテナ技術」や「Docker」が、現代のソフトウェア開発に求められるようになった理由を解説します。

矢野哲朗 棚田美寿希 , スタイルズ（2019 年 1 月 29 日）

近年、デジタルトランスフォーメーション（DX）が進展し、ビジネスを成功させるには、ソフトウェアやサービスをいかにスピーディーに開発するか、どのように効率良くシステムを運用するかが重要となってきました。その背景は、市場の動向やユーザーニーズに対して年単位でシステムを開発、更新するのではなく、もっと小さい間隔で少しずつ開発、進化させていく方が、変更の影響と失敗を極小化できるという考え方が広まってきたことにあります。

そのような開発の高速化とサービスの進化を実現するためには、短期間で要件定義からリリースまでを反復させるアジャイル開発の導入や、サービスの停止時間をできる限り削減しながら、継続的にサービスを提供、運用することが求められます。

本連載は、上記で挙げたような取り組みを実現するためのテクノロジーとして、「コンテナ技術」や、コンテナ型アプリケーション（以下、アプリ）の実行環境「Docker」、大量のコンテナ管理や負荷分散を実現する「Kubernetes」を分かりやすく解説します。「コンテナって話題になるけど何ができるか分からない」という経営者や IT 管理部門の皆さまが、コンテナ技術や関連ツールを利用したソフトウェア開発をイメージできるようになり、自社の DX について考えるきっかけになれば幸いです。

コンテナ技術が登場するまでの変遷

そもそも、なぜ、コンテナ技術や、Docker、Kubernetes に注目が集まっているのでしょうか。コンテナ技術や、Docker、Kubernetes を解説する前に、それらが生まれる前の技術的なトレンドを振り返ってみましょう。

もともと、仕事の効率化や、知的欲求を達成するために、目的に応じた専用のコンピュータ（ビジネスコンピュータ）を必要とする時代がありました。専用コンピュータの時代です。

その後、技術が進歩して、CPU の値段が安くなり、汎用的にさまざまな用途で使える小規模のコンピュータや、小規模なサーバ群が簡単に手に入る時代となりました。PC やサーバの時代です。

→目次に戻る

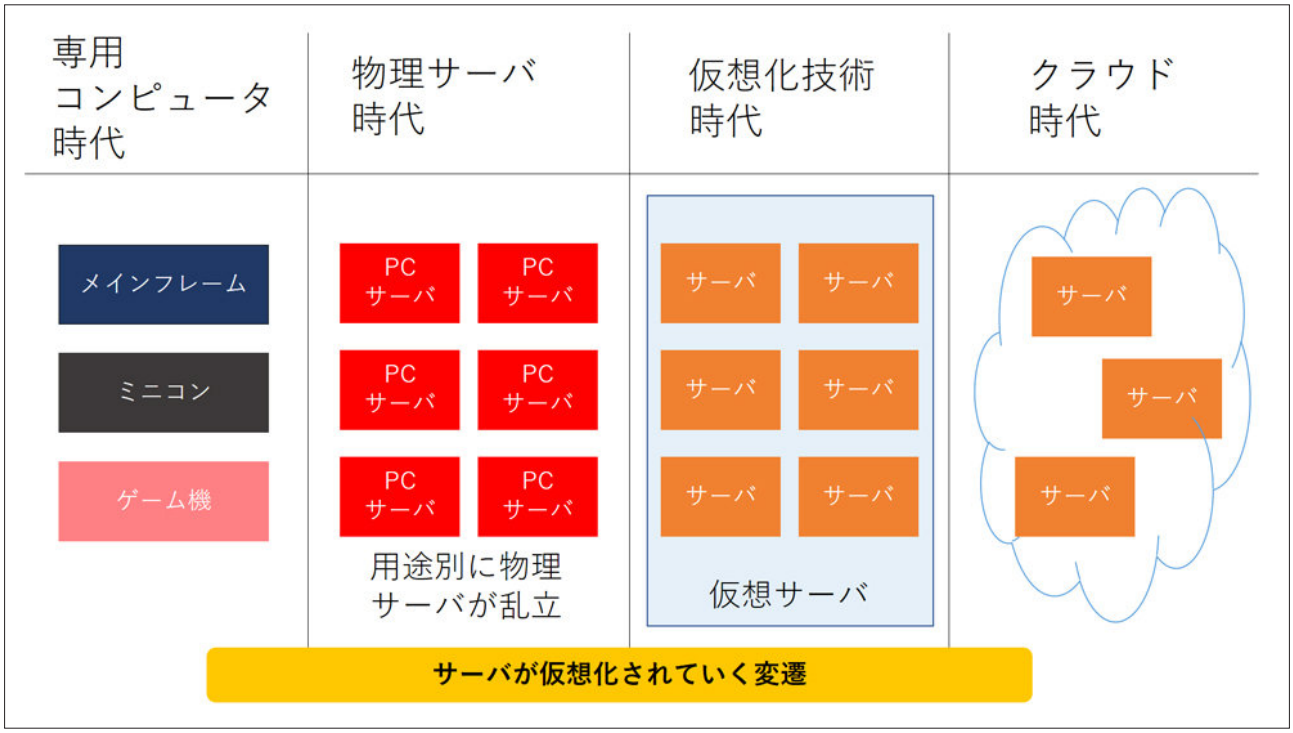
しかし、小規模な PC やサーバが増えたことにより、管理コストがかかるようになりました。また、演算処理技術の進歩により、1 つのシステムを 1 つのサーバで運用するだけでは、CPU の計算資源が余ってしまい、サーバのリソースをうまく活用できない状況が起きました。

そこで、1 つのサーバ内で、複数のシステムを扱える技術が登場します。仮想化技術の時代です。

1 つの物理サーバで、複数の仮想サーバを立ち上げてリソースを無駄なく活用できる仮想化技術は、自分で好きなようにシステム構成を変更できるメリットもあります。一方で、複数の仮想マシンを稼働させるために高性能な物理サーバを用意したり、障害に備えて物理サーバを冗長化させたりする必要がありました。

そういった仮想化技術の課題と「もう少し効率的に業務を遂行したい」「必要なときにだけ、設定済みの環境を確保したい」という開発者、運用管理者の要求を満たすために「クラウド」に注目が集まります。システムを構築する際に、クラウドを活用することで、必要なときに必要なサーバ環境（リソース）を利用するだけで済むようになりました。

しかし、仮想化技術からクラウドに進化しても、物理サーバまたは仮想サーバに入っている OS を考慮しながら、ソフトウェアやアプリを開発、運用しなければならない状況は変わりませんでした。



クラウド技術が登場するまでの変遷

→目次に戻る

クラウド時代になってからのソフトウェア開発の現状

さて、ここでいったん、クラウド時代になった昨今のソフトウェア開発の現状を整理します。

基本的にソフトウェア開発は、ユーザーから新規開発や既存機能の拡張などのリクエストがあり、そのリクエストに沿って開発の詳細を定義するところから始まります。そして、新たなソースコードを追加したり、既存のさまざまなソースコードを改修したり、連携させたりしていきます。

そのようなソフトウェア開発現場では、下記の 4 種類の環境で開発が進められます。

- 開発者の環境
- テスト担当者の環境（テスト環境／検証環境）
- ステージング環境
- 本番稼働環境

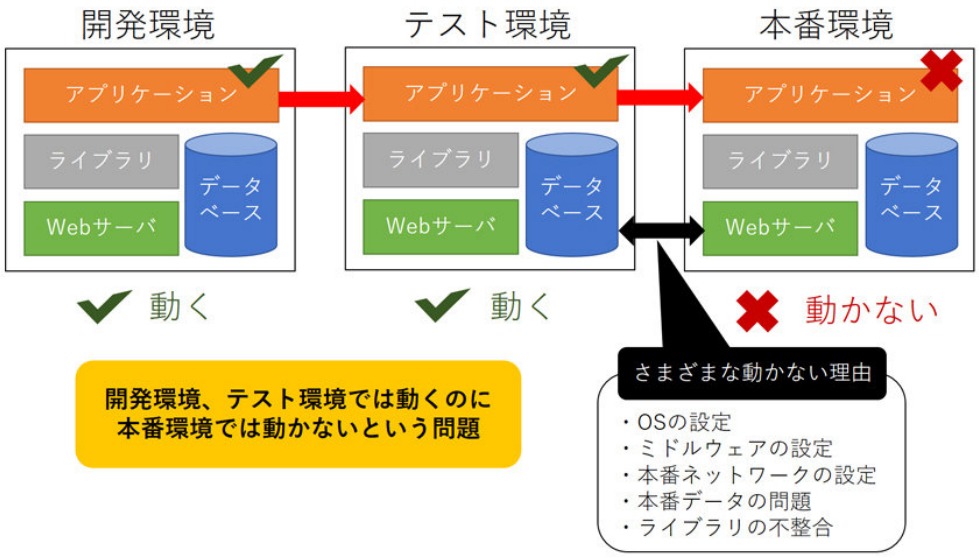
開発者の環境は、実際にソースコードを追加、修正する人自身が、ソースコードの挙動を確認する環境です。テスト担当者は、テスト環境を使って開発者のソースコードを別の観点からテストします。本稿では、この 2 つを基本的に「開発環境」とします。「ステージング環境」とは、本番前の顧客受け入れテスト用の環境です。

これらの環境において、開発者と運用管理者、それぞれの視点で課題があります。

開発者視点での課題

まず、新しくソフトウェア開発を始める際に、開発環境の構築に時間がかかってしまうという課題があります。

次に、開発環境でシステムの挙動に問題がないことを確認していても、本番環境で問題が発生することがあります。



本番環境だけソフトウェアが動作しない問題

→目次に戻る

システムが動作しない理由は、OS、ミドルウェア、ネットワーク、データ、ライブラリなど、動作環境を構成するさまざまな要素にあります。どのプロジェクトにも通用する解決方法はありません。開発者は、本当に本番環境でソフトウェアがきちんと動くか、他ソフトウェアと相互作用するような不具合はないかなど、念には念を入れて検証してからリリースする必要があります。

さらに、「開発環境はオンプレミスで、本番環境はクラウド」と、環境ごとに構成が違う場合、その検証が大変という課題もあります。

●運用管理者視点での課題

運用管理者の視点で見たとき、既に稼働しているソフトウェアやアプリを更新することは、非常に慎重にならなければならない一方で、頻繁なリリースに対応しなければならないという課題があります。

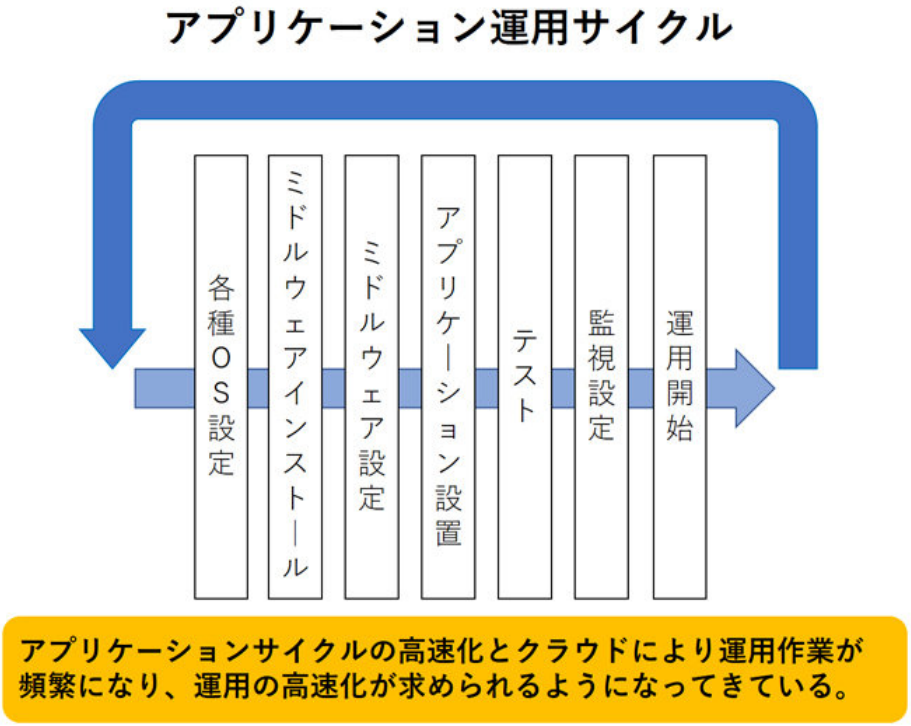
本番稼働サーバにおける運用管理者の業務は、以下のようなものです。

【構築フェーズ】

- ハードウェア調達、OS インストール（クラウドを使う場合は、クラウドベンダーの選定、IaaS でインスタンスを立てるなど）
- 各種 OS 設定
- 各種ミドルウェアインストール
- 各種ミドルウェア設定
- 稼働監視設定
- 稼働監視／運用

【運用フェーズ】

- 設定変更
- ログ監視
- ミドルウェア更新
- アプリ更新



アプリ運用サイクル

新しくサーバを構築するのは、データが入っていないので、そこまで神経質になる作業ではありません。問題は、既に運用が始まっているサーバの更新作業です。構築期間よりも運用期間の方が長いサーバの場合、更新作業にはさまざまな制約が伴います。例えば、以下のような制約です。

- サービスの利用が停止しないこと、もしくは最短の時間であること
- 利用者の少ない時間帯であること
- 再開後に、データ紛失など利用者に不便なことが起こらないこと

会社によっては、サーバの更新のために必要なコマンドを記した工程表を作成して、作業内容を全てリストアップしたものを一つ一つ手作業で 2 人以上のダブルチェックを行っている企業も多いのではないでしょうか。

●クラウド時代になってからのソフトウェア開発における課題まとめ

開発者視点と運用管理者視点で、それぞれの課題を見てきました。

開発者やサービスを提供する責任者は、より良いサービスを提供するために反復して機能の追加や改善を続けていきます。自分たちが作ったソフトウェアを早く本番で稼働させ、さらなるフィードバックが欲しいので、本番環境に早くソフトウェアをリリースしたいと考えます。

運用側は、サービスを停止せず、ソフトウェアを提供し続けることを重視します。開発者がリリースするソフトウェアに不具合があるかどうか、頻繁に検証する作業が必要になってきます。

→目次に戻る

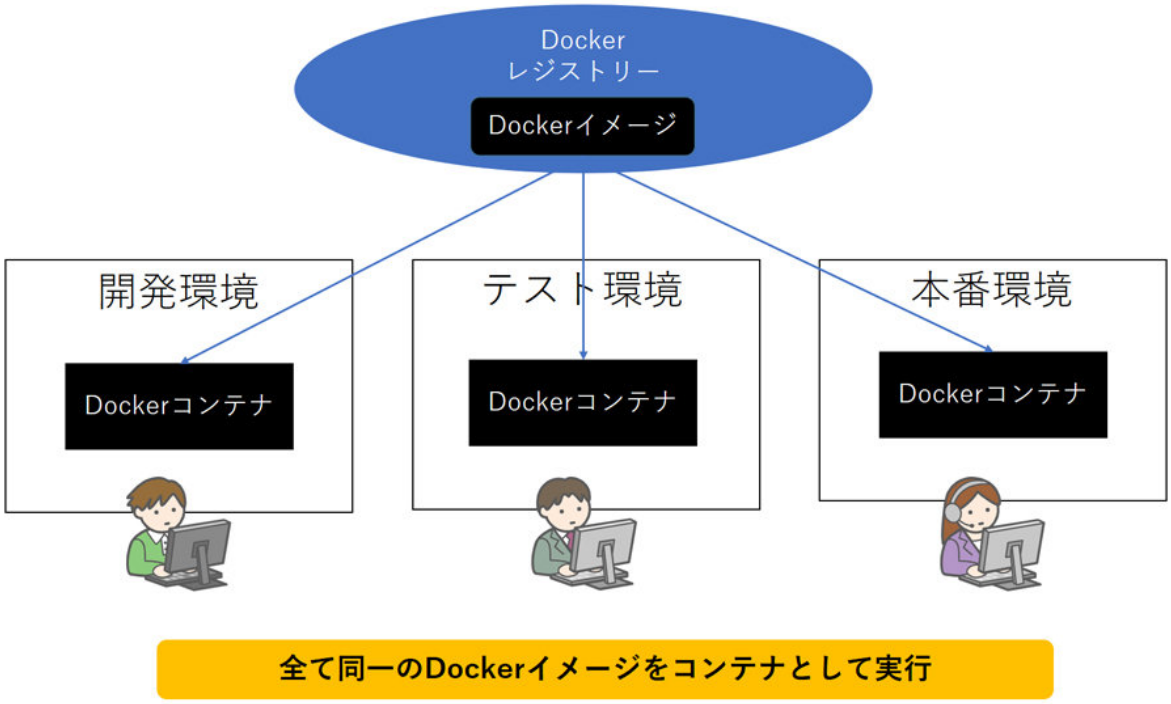
このように、開発者や運用管理者でそれぞれ求められていることが異なります。もし、開発スピードを短縮できたり、何かあっても完全にアプリのバージョンを切り戻せたりすれば、これらの課題を解決できます。筆者は、こうしたソフトウェアの開発、運用における課題を解決する最適解として生まれてきたのが、コンテナ技術だと考えています。

コンテナ技術と Docker

コンテナ技術は、実行環境を他のプロセスから隔離し、その中でアプリを動作させる技術です。コンテナが利用する名前空間やリソースは他のプロセスやコンテナからは隔離されて、それぞれ固有の設定を持てるようになります。そのため、コンテナに構築されたアプリからは、独立したコンピュータでアプリが動作しているように見えます。

コンテナ技術を用いれば、異なるサーバであっても、簡単に同一構成の開発環境や本番環境を構築できます。もちろん、何もせずにそういう構成が勝手にできるわけではありません。そのための環境が必要です。その環境の一つが、コンテナ型アプリ実行環境の Docker です。

Docker では、本番環境や開発環境の構成を「Docker イメージ」として保存できます。開発者は、Docker レジストリーから Docker イメージを取得することで、構成済みの環境を再現できます。



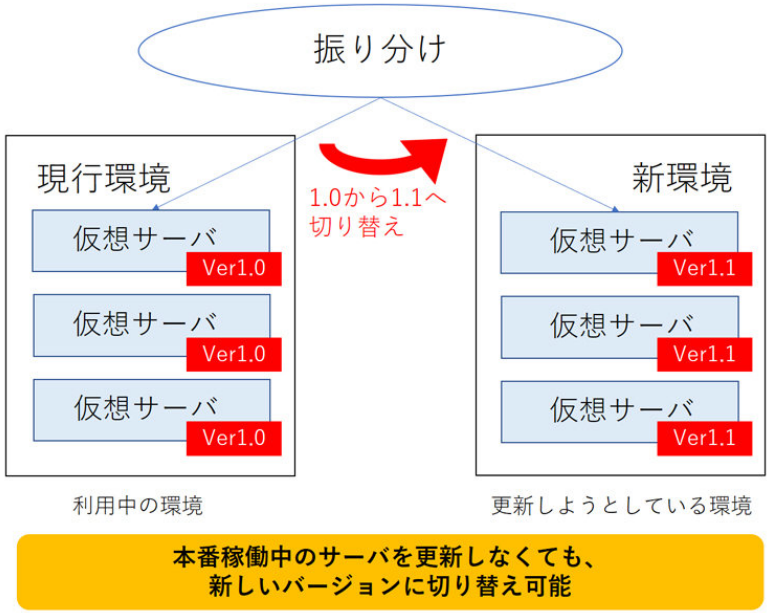
Docker イメージを用いて構成済みの環境を再現できる Docker

→目次に戻る

コンテナ技術や Docker を利用するメリット

コンテナ技術や Docker を利用するメリットについて、仮想化技術を用いて環境を構築した例で考えてみます。

バージョン 1.0 の仮想サーバ環境を完全にコピーして、事前に修正や更新、アップデートを行ったバージョン 1.1 の仮想サーバ環境を用意します。アプリを更新する際は、古い仮想マシンから新しい仮想マシンに切り替えます。元に戻す場合は、古い仮想マシンに切り替えれば、元に戻ります。



本番稼働中のサーバを更新しなくても、新しいバージョンに切り替え可能

こうすれば、本番稼働しているサーバを手順書などで更新することなく、事前にテストして動作を確認済みの新環境へ切り替えることができるはずです。

クラウドの登場後、こういった方法で新しいアプリをデプロイしたり、環境を更新したりすることが増えてきました。この移行例は、クラウドで柔軟なサーバリソースの確保ができるようになってきたから可能になった方法です。

では、コンテナだとどうでしょう。コンテナは、仮想マシンよりももっと小さい単位で、アプリを分離できます。仮想サーバの場合、1 つの仮想サーバに対して OS が必要ですが、コンテナでは不要になるため、OS の管理が不要になってセキュリティのリスクを減らすこともできます。また、上記のような仮想サーバを用いた移行例は、クラウド環境や潤沢な仮想環境でしか実現できませんが、コンテナであればオンプレミスでも同様の環境を再現でき、必要なリソースも少なく済みます。

従来のサーバ仮想化とコンテナ技術の違いについての詳細は、下記記事も参照してください。

- Docker とは

サーバでアプリを更新する作業を例えると

サーバでアプリを管理したことがない人には、Docker のメリットが分かりにくいかもしれません。そこで、「アプリ更新」という作業を、「デスクトップのアプリを更新する」例で考えてみましょう。

あるアプリがあるとします。このアプリの現在のバージョンは 1.0 です。普通、デスクトップ OS で利用中のアプリを更新するには、新しいアプリのインストーラーを入手して実行します。そうすれば新しいバージョンがインストールされ、利用できるようになります。新しいバージョンを 1.1 としておきます。

一般的なアプリのインストールと更新は、上記の通りで何も困ることはありません。しかし裏では、アプリを更新する際、新しいアプリに必要なライブラリのインストールや更新、設定が行われています。もし、アプリを更新するインストーラーがないとしたらどうでしょうか？　新しいバージョンのアプリを利用するために、アプリを手動で入れ替え、必要なライブラリを適切なフォルダに設置し、レジストリーや設定ファイルを自分で書き換えないといけないかもしれません。デスクトップアプリのインストーラーは、そういったことを実行してくれているのです。

Docker の環境では、Docker イメージに必要なアプリとライブラリの構成が記述され、Docker イメージを基に作成された Docker コンテナ（コンテナイメージといいます）にアプリやライブラリが含まれます。そのため Docker コンテナを実行すれば、アプリを実行できるのです。

では、アプリが更新されたときにバージョン 1.0 にあった機能が次のバージョン 1.1 では利用できなくなった場合は、皆さんならどうしますか？　当然、1.1 にはアップデートせず、1.0 を使い続けることになるでしょう。しかし、それでは 1.1 以降に追加される機能が利用できません。コンテナ技術の場合は、両方のバージョンが独立して存在できるので、複数のバージョンを同時に利用できます（残念ながら現在の Windows OS ではコンテナは実行できますが、デスクトップアプリのコンテナ実行まではできていません。Microsoft には、Microsoft Application Virtualization（App-V）という技術もありますが、別途導入が必要で一般的ではありません）。

「複数のバージョンを同時に利用できる」ということは、さまざまなメリットを生み出します。

例えば、1.0 から 1.1 に移行する前に、1.0 を使いつつ、1.1 を試しに使ってみて問題がないか確認することができます。「いったん 1.1 を使い始めたが問題があった」場合は、1.0 にバージョンを戻せます。また、普段利用するのは 1.1 としておき、1.0 の機能を利用したいときのみ、1.0 のアプリを起動させるということもできます。さらに、1.1 のアプリを 2 種類用意して、A 社向け環境と B 社向け環境に構築するといったことも実現できます。

まとめ

コンテナ技術やDockerにより開発者や運用管理者の負担が減り、開発環境の構築、アプリ更新のスピードが飛躍的に向上しました。その結果、Dockerの利便性を、「本番で稼働するアプリの運用にも活用できるのではないか？」と考えた人たちがいます。Kubernetesを作り始めたGoogleの開発者たちです。さらに、コンテナ技術を活用していく上で、アプリの機能の単位を小さくする「マイクロサービス」という概念にも注目が集まっています。今後は、さらにソフトウェアの開発、運用のスピードが上がり、より一層サービスや機能の開発、進化のサイクルが短くなっていくでしょう。

次回は、Dockerをビジネスで扱う際に生まれた課題、そしてその課題を解決する「Kubernetes」を紹介します。

「Excel 手順書にさようなら」――運用管理者の不安を解消する「Kubernetes」のコンセプト

大量のコンテナ管理や負荷分散を実現する「Kubernetes」について概要から本番活用の仕方まで解説する本連載。第 2 回はコンテナ技術を本番環境で活用する際の課題を解決する「Kubernetes」と 3 つのコンセプトを解説します。

矢野哲朗 棚田美寿希 , スタイルズ（2019 年 03 月 04 日）

コンテナ技術の活用により、開発者や運用管理者の負担が減り、開発環境の構築、アプリケーション更新のスピードが飛躍的に向上しました。一方で、コンテナ技術をエンタープライズのサービス提供環境（以下、本番環境）で活用するには課題があります。

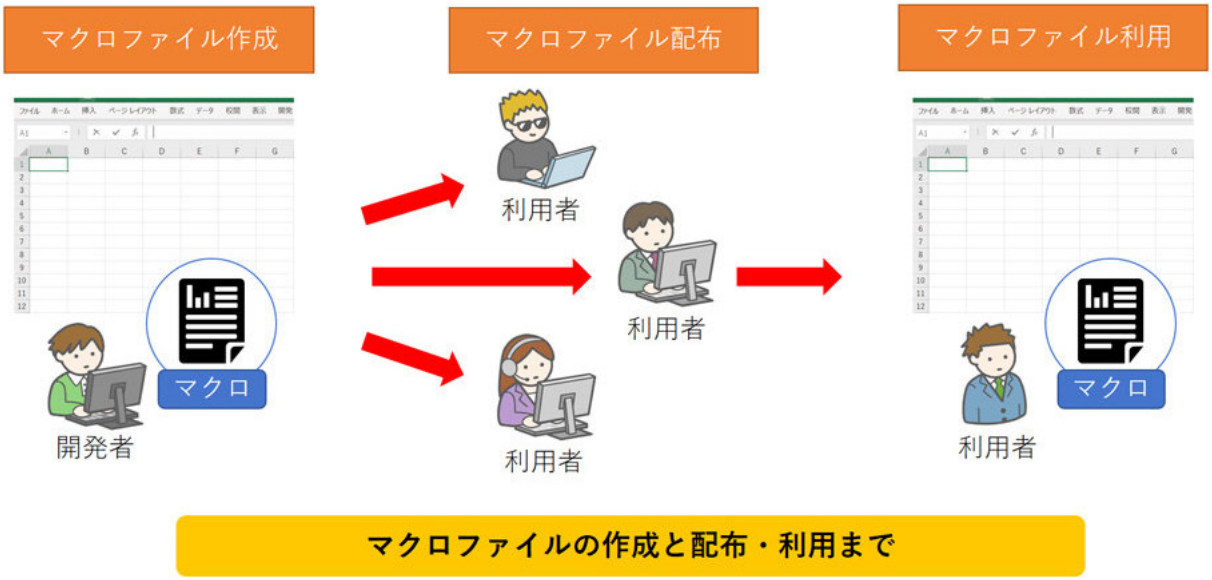
大量のコンテナ管理や負荷分散を実現する「Kubernetes」について概要から本番活用の仕方まで解説する本連載「これから始める企業のためのコンテナ実践講座」の第 2 回は、コンテナ技術を本番環境で活用する際の課題を解決する「Kubernetes」の概要と、その 3 つのコンセプトを解説します。

あらためて、コンテナ技術を利用するメリットとは

いま、ロボティクスプロセスオートメーション（RPA）がもてはやされている理由は、さまざまな業務プロセスを RPA を活用してスピードアップすることが、ビジネスに必要なものと考えられているからです。

その根底にあるのは、人力作業を削減し、IT の力で自動化することです。コンテナ技術を利用するメリットも「自動化」がキーポイントです。そして、コンテナ技術を扱うために登場した「Docker」は人の手による作業ではない「自動化された操作」を実現しました。自動化された操作とはどういうことか、スプレッドシートアプリにおけるマクロファイル（以下、マクロ）を例に説明します。

スプレッドシートアプリのマクロとは、セルに対する操作などを自動化する手法のことです。マクロを複数人で利用する場合、マクロを利用者全員に配布する必要があります。また、マクロをフォルダに配置したり、マクロが実行できるバージョンのスプレッドシートアプリを用意したりと、手作業で行う必要があります。



マクロ作成と配布、利用まで

コンテナ技術が実現した「自動化された操作」とは、マクロ作成後の配布、実行までの一連の操作のことです。マクロと一緒にスプレッドシートアプリを配布すれば、スプレッドシートアプリとマクロファイルのバージョンの不整合は解消されます。自動化された操作を実現することで、アプリケーションリリースのハードルを大きく下げ、かつ、人的作業を削減できるわけです。

現在のサーバアプリケーションの世界では、コンテナ技術を利用することで、以下のようなメリットを享受できる時代になったといえます。

- 自動で環境を構築できる
- アプリケーションを修正するたびに自動テストを実行できる（自動テストがしやすくなる）
- 自動テストによりバグを発見しやすくなる（コード品質と開発スピードの向上）

ちなみに、コンテナ技術の世界では先述した作成、配布、利用のことを「ビルド（構築）、シップ（配布）、ラン（実行）」といい、まとめて「コンテナライフサイクル」と呼びます。

コンテナライフサイクル

→目次に戻る

コンテナライフサイクルを実現するコンテナ実行環境として最も利用されているのは「Docker」ですが、昨今は「Kubernetes」というコンテナ管理ツールも話題になっています。なぜ、Kubernetes が話題になっているのでしょうか。それは、本番環境で Docker だけを利用して運用するには不便な部分があるからです。

Docker の不便な部分とは

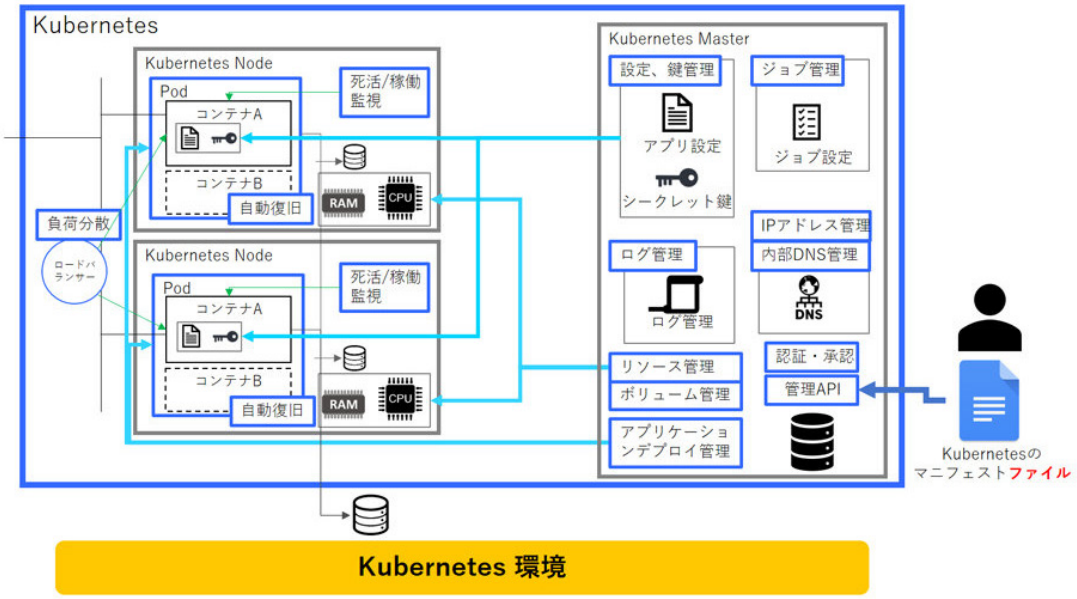
開発環境と本番環境には、大きな違いとして「サービスの停止を許容できるかどうか」「厳格な管理が必要かどうか」があります。Docker でダウンタイムのないサービスを大規模運用するには、さまざまな問題を解決しなければなりません。例えば以下のような問題です。

- Docker の永続ストレージの管理の問題
- Docker のネットワーク、ポート番号管理の問題
- コンテナでの設定ファイル、秘匿情報管理の問題
- コンテナ稼働状況管理の問題、スケジューリングの問題
- コンテナ用ホストの管理の問題

開発者が利用する点でいえば、実行環境の準備が容易で、さまざまな知見がネットで公開されている Docker は優れたツールといえます。しかし、Docker を本番環境で利用するには以上のような問題があったのです。

エンタープライズの本番環境に活用される「Kubernetes」

Kubernetes は、本番環境でコンテナ技術を利用するために Google が開発を始めたコンテナ管理ツールです。Google が提供するさまざまなサービスの運用ノウハウが、Kubernetes に注ぎ込まれています。例えば、Kubernetes には、サービスを公開した状態でさまざまな変更を加えることができる仕組みになっています。



Kubernetes 環境

→目次に戻る

Kubernetes が持つ機能は以下のようなものです。

- 複数の Kubernetes Node の管理
- コンテナのスケジューリング
- コンテナのローリングアップデート
- コンテナのサービスディスカバリとロードバランシング
- スケジューリング／オートスケジューリング
- コンテナの死活監視
- 障害時のセルフヒーリング
- リソース管理
- ストレージ管理
- 設定ファイル管理
- パスワード管理
- ネットワーク管理
- ログ管理

コンテナ技術では「永続的なデータを保存できない」「設定ファイルや秘匿情報を埋め込むことができない」といった欠点があります。そのため、サービスの継続性に致命的な問題を引き起こす可能性がありましたが、Kubernetes はそれらの欠点をカバーしつつ、本番環境にあるサービスを、コンテナのメリットを活用しながら継続できる機能を組み込んでいます。

Docker でもさまざまな欠点に対策が用意されていました。しかし、サービスを停止させない仕組みや、コンテナを統合的に管理できるという点で Kubernetes は優れています。

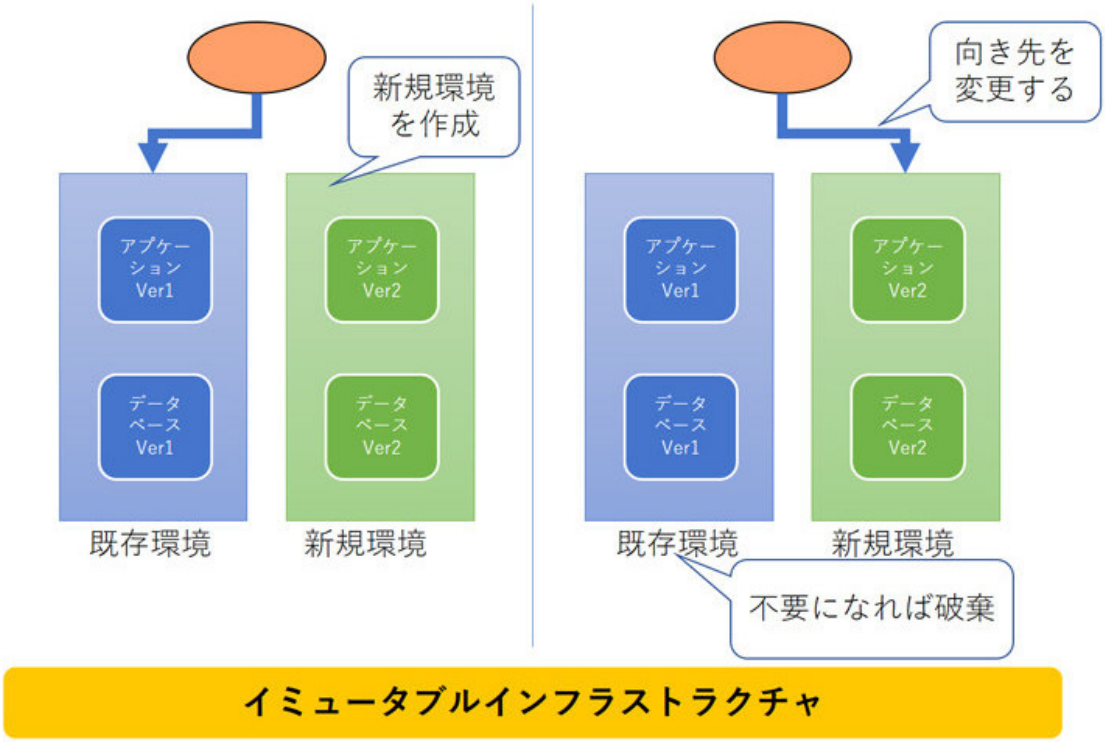
Kubernetes の特徴的なコンセプト

Kubernetes を活用する前に、これまでのインフラ管理とは違う特徴的な 3 つのコンセプトを知っておきましょう。

- イミュータブルインフラストラクチャ
- 宣言的設定
- 自己修復機能

1.「イミュータブルインフラストラクチャ」

イミュータブルインフラストラクチャ（Immutable Infrastructure）とは、「環境を変更する場合は、既存の設定を変更して、環境を更新するのではなく、設定が変更された新しい環境を用意し、古い設定の環境は捨てる」という考え方です。構築された環境は変更しないことをポリシーにします。これまでのインフラ管理の考え方からすると非常に大胆な考え方です。



イミュータブルインフラストラクチャ

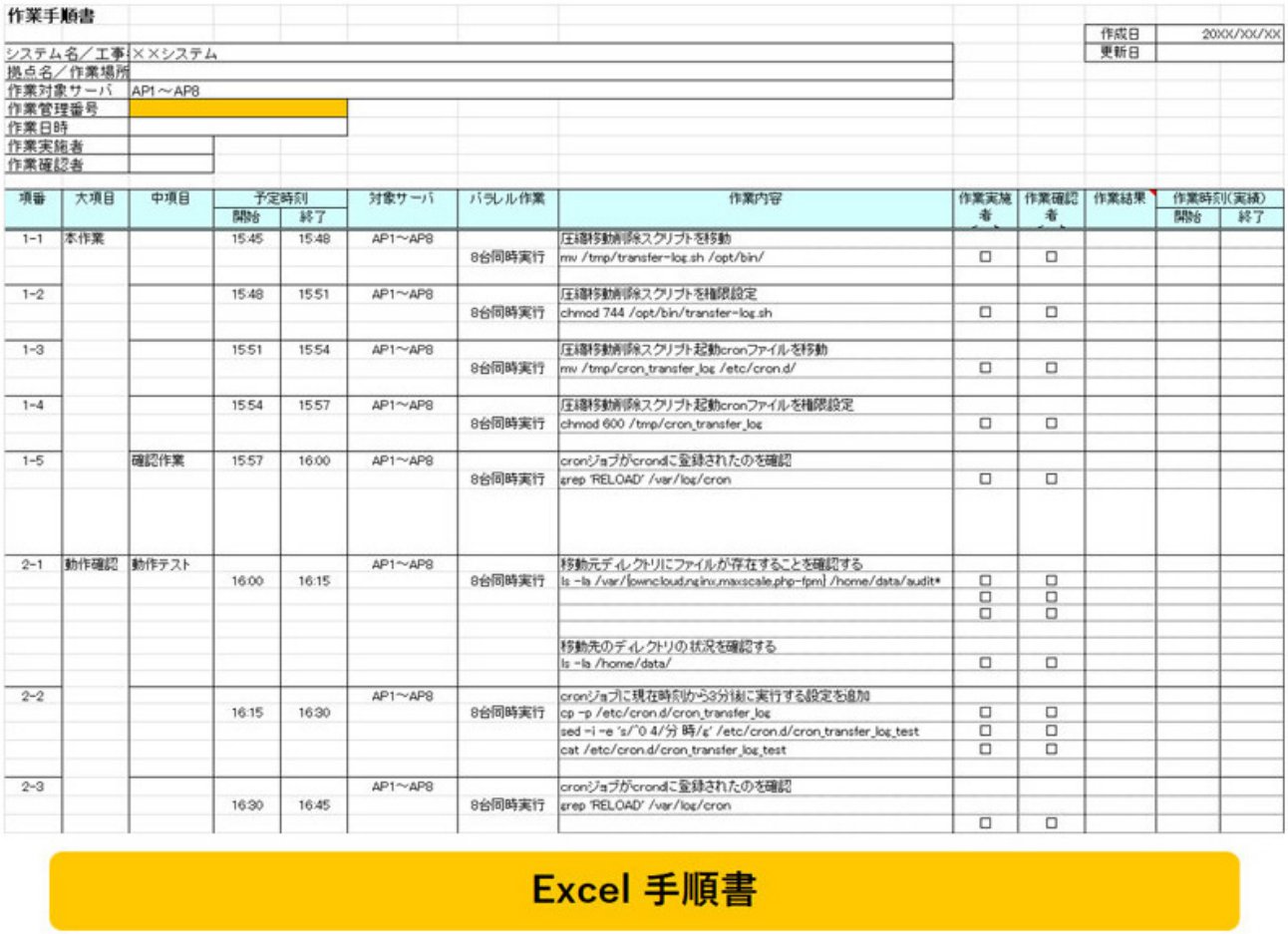
イミュータブルインフラストラクチャには大きなメリットが 2 つあります。

- 新しい環境に変更する際、既存設定との差分を把握する必要がなくなること
- 設定手順が自動化できること

既存の設定を無視し、新しい設定で環境を作り直せば「既存の設定が邪魔をして新しい設定を環境に適用できない」ということはまずあり得ません。第 1 回で「初期構築は何もない状態から作るので既存環境の影響を考える必要がなく、さほど問題なく作ることができる」と解説した通り、毎回新しく作り直せば既存環境がどうであるか気にする必要がないのです。

これは 2 つ目のメリットである設定手順の自動化にもつながります。従来のインフラ管理では、既存設定がどうなっているかを確認し、環境の更新時にミスが生まれないよう「Excel 手順書」を利用するケースが多くありました。

→目次に戻る



作業手順書

作成日	20XX/XX/XX
更新日	

システム名／工事	××システム
拠点名／作業場所	
作業対象サーバ	AP1～AP8
作業管理番号	
作業日時	
作業実施者	
作業確認者	

項番	大項目	中項目	予定時刻		対象サーバ	パラレル作業	作業内容	作業実施者	作業確認者	作業結果	作業時刻(実績)	
			開始	終了							開始	終了
1-1	本作業		15:45	15:48	AP1～AP8		圧縮移動削除スクリプトを移動					
						8台同時実行	mv /tmp/transfer-log.sh /opt/bin/	□	□			
1-2			15:48	15:51	AP1～AP8		圧縮移動削除スクリプトを権限設定					
						8台同時実行	chmod 744 /opt/bin/transfer-log.sh	□	□			
1-3			15:51	15:54	AP1～AP8		圧縮移動削除スクリプト起動cronファイルを移動					
						8台同時実行	mv /tmp/cron_transfer_log /etc/cron.d/	□	□			
1-4			15:54	15:57	AP1～AP8		圧縮移動削除スクリプト起動cronファイルを権限設定					
						8台同時実行	chmod 600 /tmp/cron_transfer_log	□	□			
1-5		確認作業	15:57	16:00	AP1～AP8		cronジョブがcrondに登録されたのを確認					
						8台同時実行	grep 'RELOAD' /var/log/cron	□	□			
2-1	動作確認	動作テスト			AP1～AP8		移動元ディレクトリにファイルが存在することを確認する					
			16:00	16:15		8台同時実行	ls -ls /var/[owncloud,nginx,maxscale,php-fpm] /home/data/audit*	□	□			
								□	□			
								□	□			
							移動先のディレクトリの状況を確認する					
							ls -ls /home/data/	□	□			
2-2					AP1～AP8		cronジョブに現在時刻から3分後に実行する設定を追加					
			16:15	16:30		8台同時実行	cp -p /etc/cron.d/cron_transfer_log	□	□			
							sed -i -e 's/^0 4/分 時/g' /etc/cron.d/cron_transfer_log_test	□	□			
							cat /etc/cron.d/cron_transfer_log_test	□	□			
2-3					AP1～AP8		cronジョブがcrondに登録されたのを確認					
			16:30	16:45		8台同時実行	grep 'RELOAD' /var/log/cron					
								□	□			

Excel 手順書

Excel 手順書

Excel 手順書は、サーバ機器の設定を事前に確認して一つずつ順番に記載したものです。この手順書を元に、作業員が運用業務を行ってきました。ミスを防ぐために複数名で実施し、実施時刻を記入して「作業実績エビデンス」として残す場合もあります。

これは「更新作業を自動化できない」という問題から利用されてきましたが、イミュータブルインフラストラクチャの「古いものは捨てて、その都度作成する」という考え方により、「自動化できない手順書での対応」というものをなくしているのです。

2.「宣言的設定」

2 つ目のコンセプトは、宣言的設定です。従来、サーバの管理には Excel のパラメーターシートなどを用いて管理していました。さまざまなミドルウェアの設定を全て Excel のシートに書き出し、一つずつそれに沿ってサーバを構築します。しかし、これにも問題があります。

Excel 手順書を用いた更新作業後、パラメーターシートが更新されないことが多々あるのです。その結果、パラメーターシートを確認してもサーバ側の設定が分からず、結局サーバでコマンドを操作して確認するという「2 重管理」が起きます。

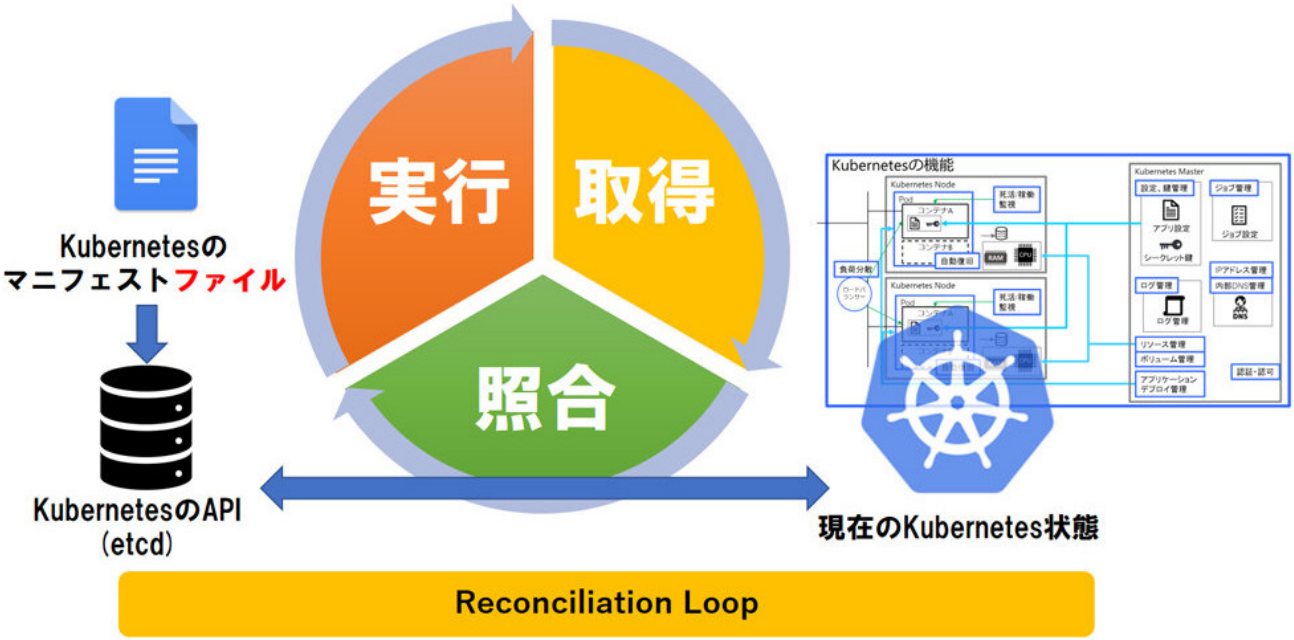
宣言的設定では、この 2 重管理を起こさないため、Kubernetes でのマニフェストファイルを基準にします。サーバ側の設定を変更する際は、このマニフェストファイルを先に修正し、Kubernetes 環境に適用します。その結果、必ず Kubernetes のマニフェストファイルの設定と Kubernetes 環境の設定が同一のものになります。もし同じ環境を再現する場合は、このマニフェストファイルを別の場所で適用すればいいのです。

3.「自己修復機能」

3 つ目のコンセプトは、自己修復機能です。自己修復機能では、マニフェストファイルに記述された「システムのあるべき姿」と、現在の Kubernetes のコンテナ環境の状態が同じ状態かどうかを監視します。差分が発生した際、既存の環境の状態を更新するのではなく、新しい状態の環境を新しく用意して「システムのあるべき姿」に近づけます。

例えばマニフェストファイルで「サーバを 3 台稼働させる」と宣言すれば、Kubernetes は「サーバが 3 台稼働している状態」を維持するように動作するわけです。

この「システムのあるべき姿」と「現在の環境の状態」の差分を常時監視することを「Reconciliation Loop」と呼びます。



Reconciliation Loop

常にサーバの設定と本番環境の状況が同じ状態かどうか監視するのは、コンピュータ的には負荷の高い動作です。従来のサーバ運用のように、設定を変えたときだけ接続し、操作する方が、無駄のない挙動になります。しかし、負荷が高かったとしても、設定と現在の環境の状態の整合性を保ち続けることで、マニフェストファイルでの設定が常に「正」であり、サーバの状態と同一なことが保証される仕組みになっています。

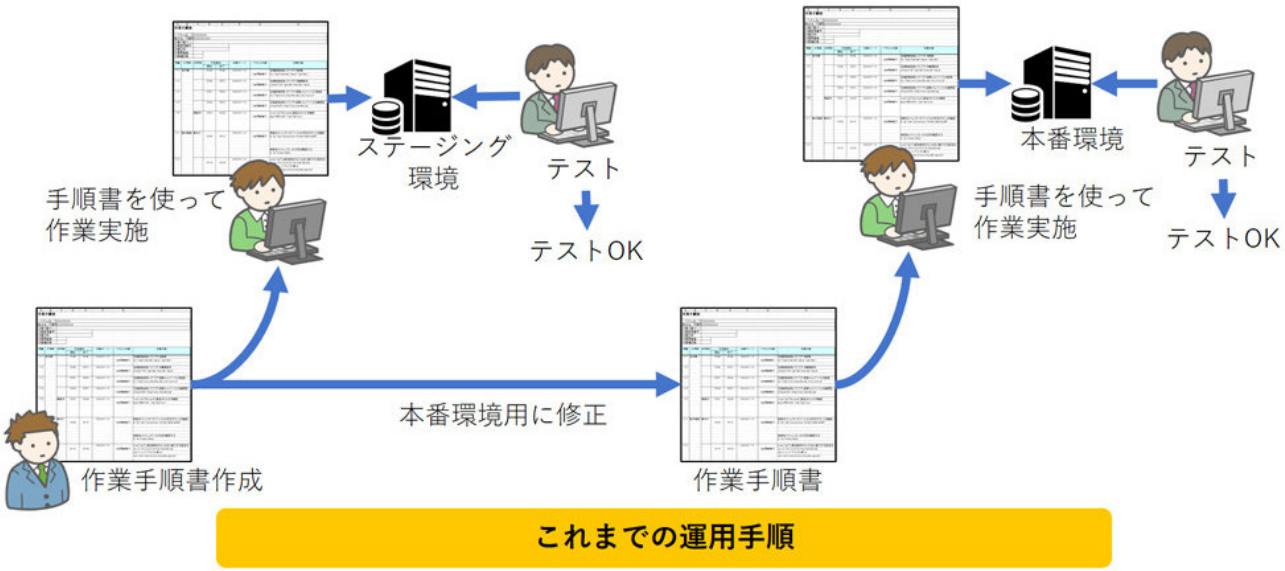
さて、Kubernetes の 3 つのコンセプトを読んで、運用管理者の方は「マニフェストファイルの設定を間違えて本番環境に適用した結果、サービスやサーバが動作しなくなったら大変だ」と想像するかもしれません。

実は確かにその通りです。しかし、本番環境にマニフェストファイルの設定を適用させる前に、その設定が正しいかどうか、別のコンテナ環境でテストすればよいのです。もし万が一間違った設定を本番環境に適用したとしても、コンテナが動かないような設定の場合には、Kubernetes が自動で停止させる機能を用意しています。

まとめ

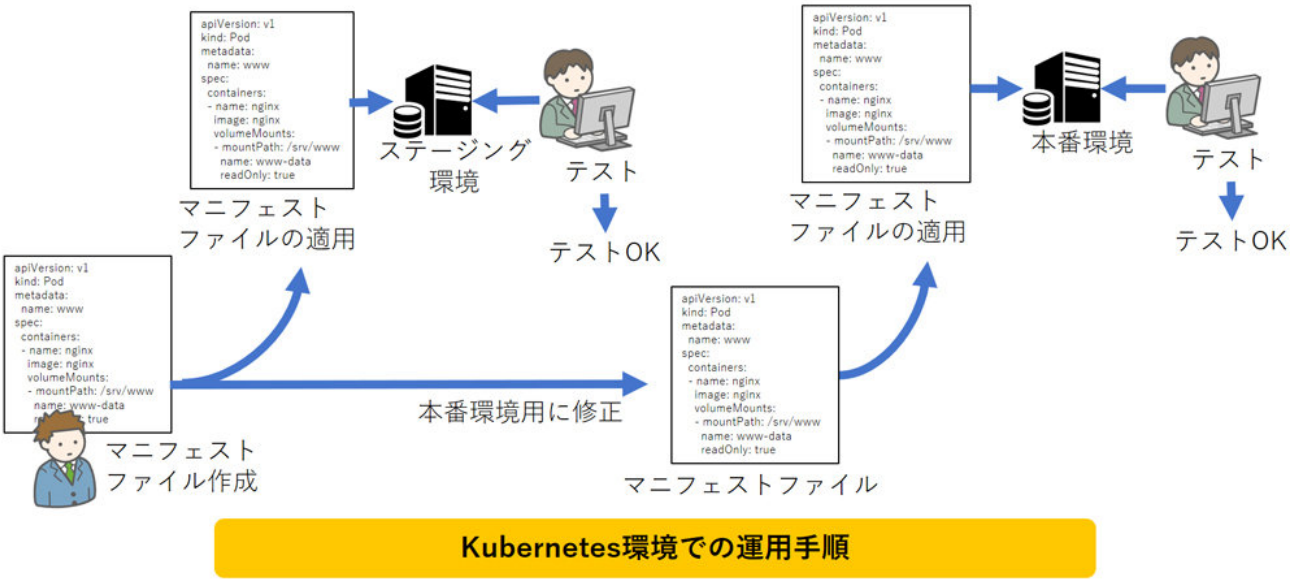
これまでの運用方法と比較して、Kubernetes を用いることで、手作業が大幅に削減され、誰でも同じように環境を構築、修正できるようになりました。しかし、このような便利な機能を使えるにもかかわらず、これまでのオンプレミスサーバと同じように手順書を作り、逐次的な作業をして差分的に運用していたらどうでしょうか？　それではせっかく便利な新しいツールに変更したのに、運用はそのままで楽になっておらず、これまでの課題が解決されないまま自動化も省力化もされないことになってしまいます。

Kubernetes は、これまでの運用を一変させ、便利に安全に利用できる管理ツールです。ぜひ、このコンセプトを把握した上で Kubernetes を使いこなし、従来の手作業でサーバ設定を変更するような運用をなくしていきましょう。



これまでの運用手順

→目次に戻る



Kubernetes 環境での運用手順

次回は、Kubernetes のアーキテクチャと主要機能の詳細、また、Kubernetes が「分散システムとしての Linux」と呼ばれている理由について解説します。

→目次に戻る

Kubernetes がクラウド界の「Linux」と呼ばれる 2 つの理由

大量のコンテナ管理や負荷分散を実現する「Kubernetes」について概要から本番活用の仕方まで解説する「これから始める企業のためのコンテナ実践講座」第 3 回は、Kubernetes がクラウド界の「Linux」と呼ばれている理由とともに、Kubernetes 内部の仕組み、機能を紹介します。

矢野哲朗，スタイルズ（2019 年 04 月 08 日）

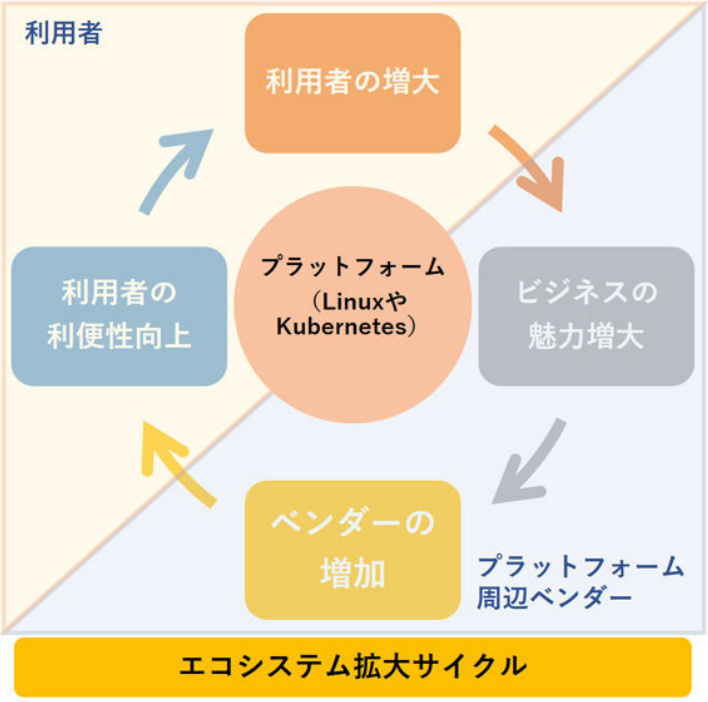
大量のコンテナ管理や負荷分散を実現する「Kubernetes」について、概要から本番活用の仕方まで解説する本連載「これから始める企業のためのコンテナ実践講座」第 2 回では、エンタープライズのサービス提供環境（以下、本番環境）でコンテナ技術を活用する際の課題を解決する Kubernetes の概要と、3 つのコンセプトを紹介しました。

第 3 回は、Kubernetes が何を実現するのかについてや、Kubernetes が「クラウド界の Linux」と呼ばれる理由とともに、Kubernetes 内部の仕組み、機能を紹介します。

Kubernetes がクラウド界の Linux と呼ばれる理由

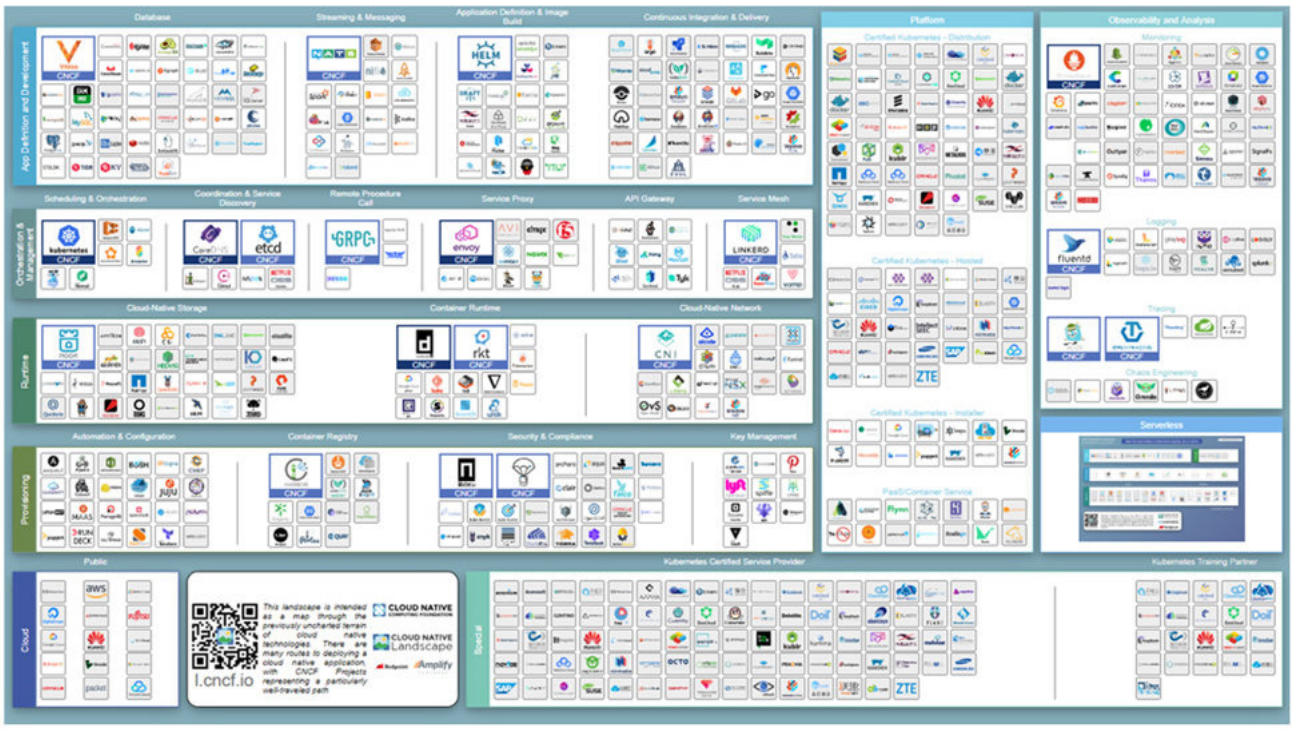
なぜ Kubernetes はクラウド界の Linux と呼ばれているのでしょうか。その理由は、2 つあります。1 つ目は「Linux のようなエコシステムとコミュニティーを持っていること」、2 つ目は「Linux のような機能を持っていること」です。まずは 1 つ目について見ていきましょう。

Linux のエコシステムとは、「さまざまなベンダーが Linux というプラットフォームを強化した結果、Linux への魅力が増して、Linux 利用者が増え、他のベンダーも Linux に関わっていく」というように、さまざまなベンダーが Linux を中心に考えるようになるサイクルのことです。



エコシステム拡大サイクル

Kubernetes の場合はどうでしょうか？　Kubernetes の管理団体である「Cloud Native Computing Foundation」（以下、CNCF）では、Kubernetes を拡張するベンダーや、関連ツールを開発するベンダーを「The Cloud Native Interactive Landscape」としてまとめています。これを見ると多くのベンダーが Kubernetes をプラットフォームとして、周辺の機能やサービスを整備していることが分かります。



Kubernetesを取り巻くソフトウェア群
（出典：「The Cloud Native Interactive Landscape」）

Kubernetes を取り巻くソフトウェア群（出典：「The Cloud Native Interactive Landscape」）

→目次に戻る

これらのベンダー、サービスにより Kubernetes の魅力が増した結果、Kubernetes の利用者が増えてきています。それを裏付けるように、CNCF が主催する Kubernetes カンファレンス「KubeCon + CloudNativeCon」では、2016 年の参加者数が 1139 人だったのに対し、2017 年に 4212 人、2018 年には 8000 人 (キャンセル待ち 1000 人) となりました。また、Kubernetes を取り上げるメディアや書籍も増えており、今後、興味を持った企業によって Kubernetes に関するさまざまなサービスが登場することは間違いないでしょう。

これらのことから、Kubernetes は Linux のようなエコシステム、コミュニティーを持っているといえます。

Kubernetes と Linux はどう似ているか？

Kubernetes が持つ機能を Linux と比較しながら紹介する前に、Linux（Linux ディストリビューション）が何をするのか、改めて振り返ってみましょう。Linux はオペレーティングシステム (以下、OS) であるということは分かりますが、本当は何をしているのでしょうか。

Linux OS の根幹である「Linux カーネル」では、下記を管理します。

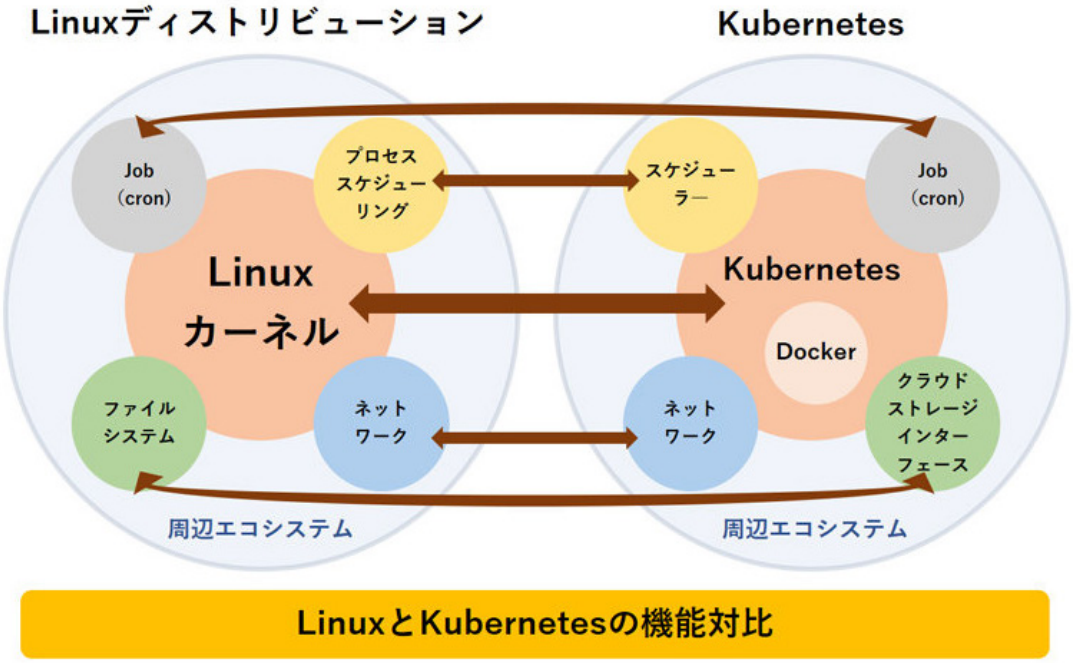
- プロセス (ライフサイクル)
- 名前空間
- メモリ
- プロセススケジューリング
- 割り込み
- デバイス
- ファイルシステム
- ネットワーク
- ネットワークセキュリティ

Linux OS 全体としての「Linux ディストリビューション」では下記を管理します。

- ユーザー
- 承認
- 認可
- ネットワーク
- ログ
- セキュリティ
- ジョブ

- UI ／ GUI
- DNS
- パッケージ (rpm, deb)

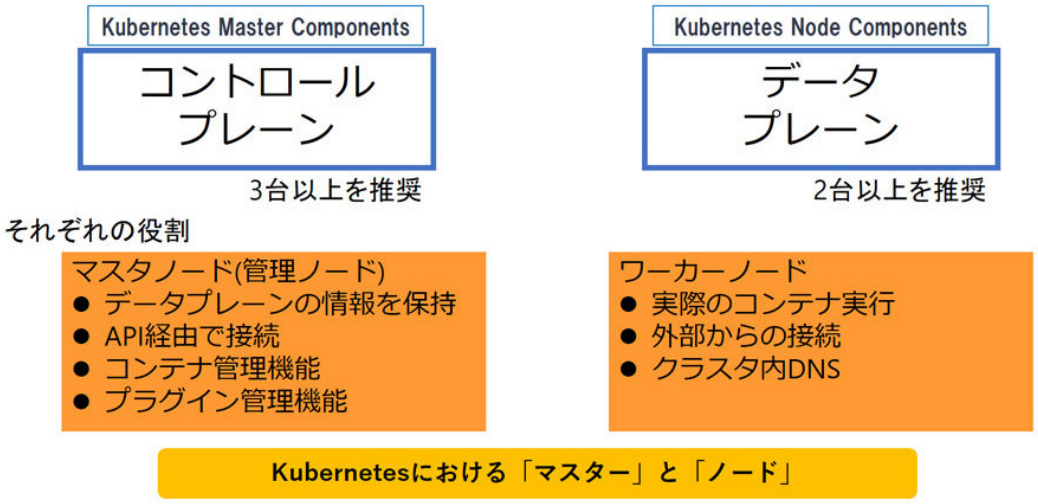
Linux カーネルの機能は、ホスト単体での機能ですが、Kubernetes は複数のホストを束ねたクラスタで同じような機能を用意しています。ざっくり図にしたのが下記です。



Linux と Kubernetes の機能対比

Kubernetes 内の役割分担

それでは、Kubernetes にはどのような機能が備わっているのでしょうか？　機能について紹介する前に、Kubernetes にある大きな 2 つの役割「Kubernetes Master components」（以下、マスター）と「Kubernetes Node components」（以下、ノード）を紹介します。



Kubernetes における「マスター」と「ノード」

→目次に戻る

Kubernetes Master Components

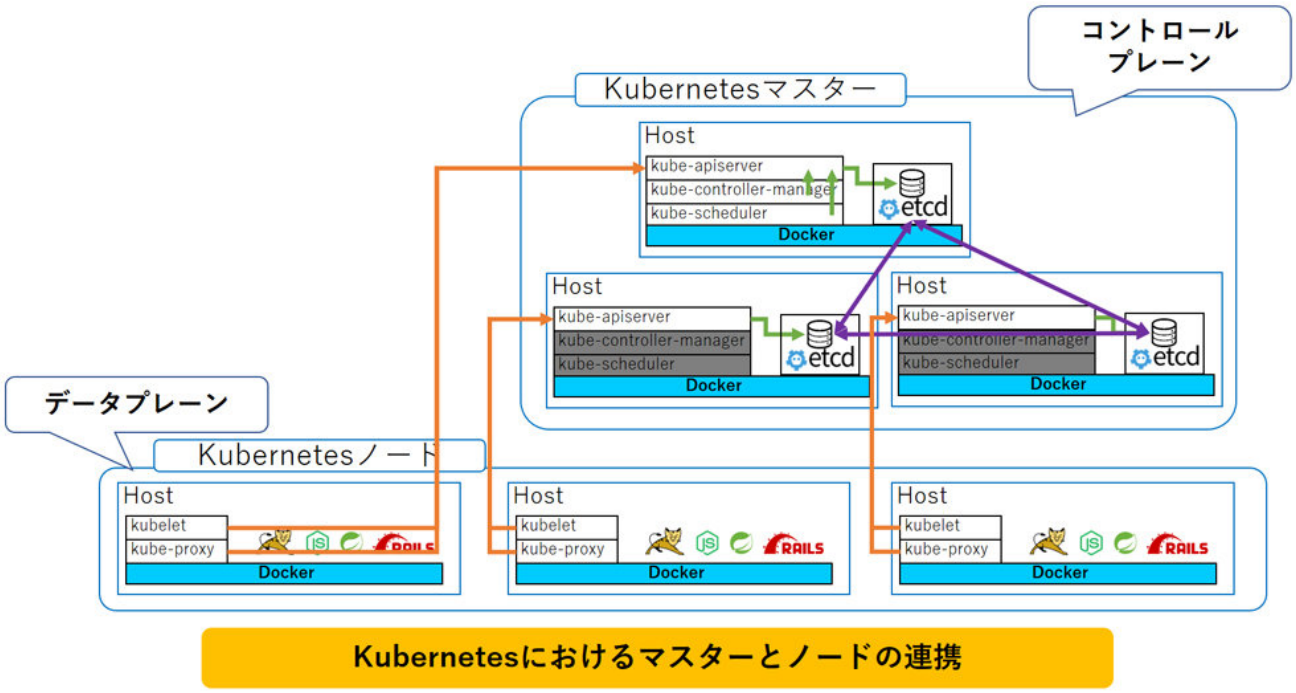
マスターでは、コンテナを管理する役割が備わっています。こういった役割を持つコンポーネントは一般的に「コントロールプレーン」と呼ばれています。

Kubernetes では最低 1 台からマスターを構成できますが、3 台以上のサーバでマスターを構成することを推奨しています。もし、マスターを構築したサーバが 1 台の環境でマシンが落ちた場合、コンテナを管理できなくなってしまうためです。

Kubernetes Node Components

ノードは、マスターの下で指示されたコンテナを実行する役割が備わっています。こういった機能を持つコンポーネントを一般的に「データプレーン」といいます。

ノードもマスターと同様、最低 1 台から構成できますが、Kubernetes では 2 台以上のサーバでノードを構成することを推奨しています。



Kubernetes におけるマスターとノードの連携

マスターとノードのまとめ

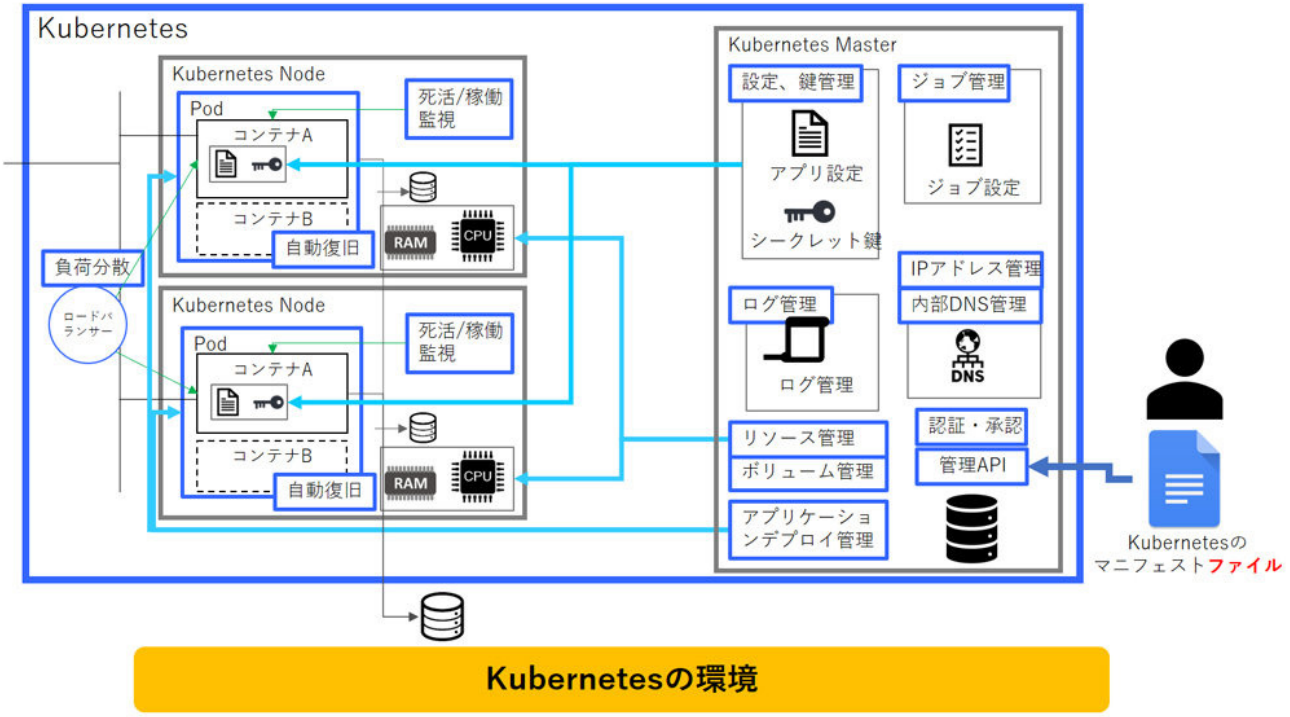
マスターとノードをまとめると、推奨要件で Kubernetes を運用するには 5 台以上のサーバが必要です。サービスを運用するためには、さらに多くのサーバが必要になります。検証段階で、5 台以上の物理サーバを用意することは難しいため、利用している企業は仮想マシンで実行したり、下記のような工夫を行ったりしているようです。

- マスターとノードを両方 1 台で済ませる
- 社内仮想環境でマスターを 1 台、ノードを複数台用意する
- 社内仮想環境でノードとマスターを同居させて 3 台用意する
- クラウドサーバでマスターを 3 台とノードを複数台用意する
- クラウドサービスのマスターを利用してノードを複数台用意する

この部分については、次回以降解説します。

Kubernetes の構成

Kubernetes 内部の役割分担について説明しましたが、Kubernetes の基本的な機能は、マスターにあるサービスで説明できます。連載第 2 回で紹介した Kubernetes を活用した場合のサーバ機能についての図を再度掲載します。

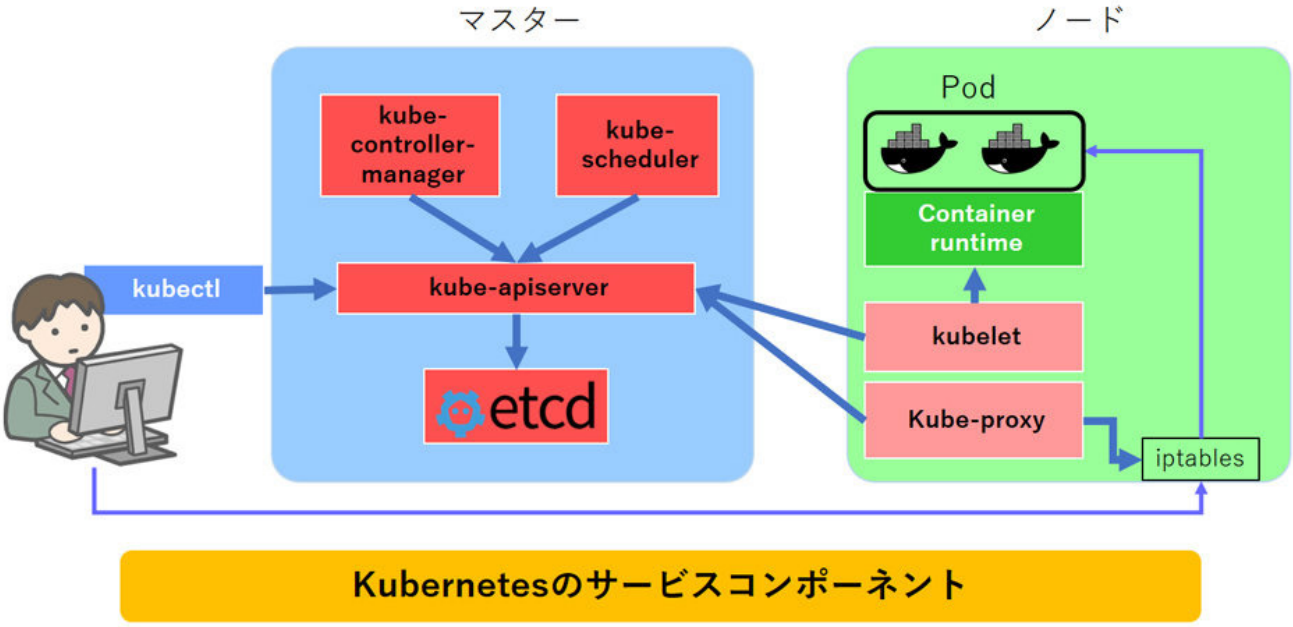


Kubernetes の環境

Kubernetes を利用する際に必要な上記の基本機能は、下記 4 つのサービスコンポーネントで構成されています。

- etcd
- kube-apiserver
- kube-scheduler
- kube-controller-manager

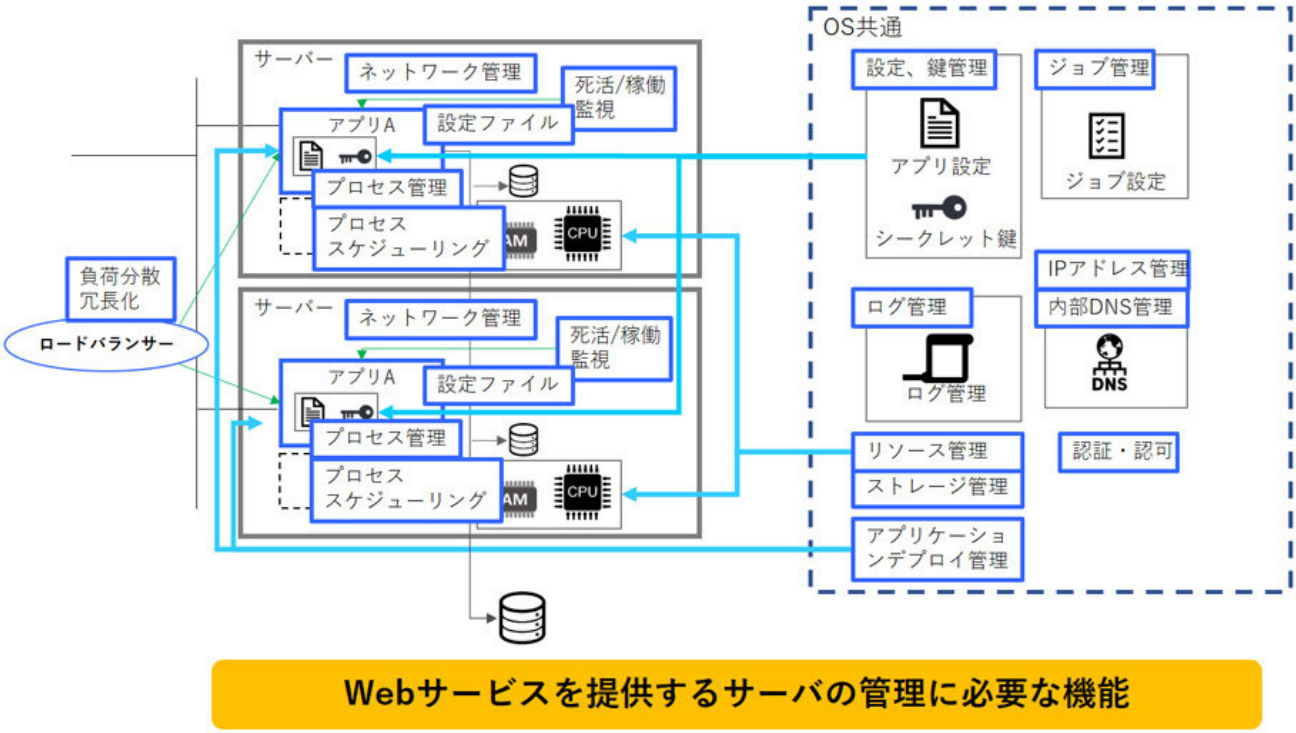
→目次に戻る



Kubernetes のサービスコンポーネント

それ以外の機能は、別途その機能を持つコンテナを実行したり、上記の各サービスで呼び出される定義やコントローラーを実装したりすることで提供されています。なるべく基本要素を増やさず、シンプルな構成でさまざまな要素に機能しようとしているわけです。

サービスコンポーネントが提供している Kubernetes の各種機能と Linux での機能を比べてみましょう。さらに、Web サービスを提供するサーバの管理には以下のような機能が必要です。Kubernetes にはどのような機能が用意されているかも含めて説明します。



Web サービスを提供するサーバの管理に必要な機能

→目次に戻る

プロセス管理

Kubernetes では「プロセスの管理＝コンテナの管理」になります。Linux がプロセスを終了するまで管理するように、Kubernetes はコンテナが起動してから終了するまでを管理できます。

プロセスのスケジューリング

Kubernetes では、コンテナをどこのホストでいつ実行するか、実行優先度をどうするかなどを管理できます。

リソース管理

Kubernetes では、コンテナ 1 つに割り当てる CPU やメモリの上限と下限を指定して管理できます。

ストレージ管理

Kubernetes の場合、コンテナが永続的なものではないため、OS に付いている「ディスク」のような概念はありません。しかし、それではサービスを運用する際の永続データ領域や一時保存領域などがなくなってしまうため、「Volume」「PersistentVolume」といった機構を用意して、永続データや一次保存領域を確保できます。

ユーザー管理

Linux のログインとは異なり、Kubernetes では、コンテナを操作できるかどうかの権限を用意しています。もっと詳細に説明したいところですが、ここでは割愛します。

ネットワーク管理

Kubernetes では、コンテナを「Pod」というオブジェクトで管理します。各 Pod に対して IP アドレスを割り当て、Linux の「iptables」のように接続許可、拒否ができます。

バッチ動作管理

Kubernetes では、Linux の「Cron」のように定期的な起動を行う「CronJob」とタスクの実行回数と並列数を指定して実行する「Job」があります。

パッケージ管理

Kubernetes では、コンテナ自体がパッケージです。そのコンテナを起動するマニフェストを汎用化して利用しやすくする「Helm」というパッケージマネージャーがスタンダードとなっています。

ログ管理

Kubernetes では、コンテナから出力されるログを集約する仕組みがあります。基本的に標準出力と標準エラー出力を収集するため、コンテナが稼働しているときに出力された標準出力と標準エラー出力のログしか収集できません。そのため、ログ集約ツールの「Fluentd」を活用したり、エージェントを立ち上げてログファイルを転送したりするなど、コンテナの状態と関係なくログを収集する仕組みを各自で構築することが多いようです。

設定ファイル

先述した主要機能をどのように動作させるかを示す設定ファイルは、Kubernetes の場合、Docker コンテナイメージではなく「ConfigMap」で外部に持つことができます。

サービスを提供するために必要な機能

以上のように、Kubernetes は Linux と似たような機構を持っています。続いて、サービスを提供するために必要な機能は、どのようなものかを見ていきましょう。

サービスを実現するためのアプリケーション

Kubernetes では「コンテナイメージ」がサービスを実現するためのアプリケーションに当たります。

アプリケーションを実行するための設定ファイルや環境変数

Kubernetes では、Kubernetes 自身の設定ファイルと同様に「ConfigMap」で、アプリケーション用の設定や環境変数を管理できます。

アプリケーションで利用されるデータ

Kubernetes では外部ストレージをマウントすることができます。

アプリケーションの稼働監視

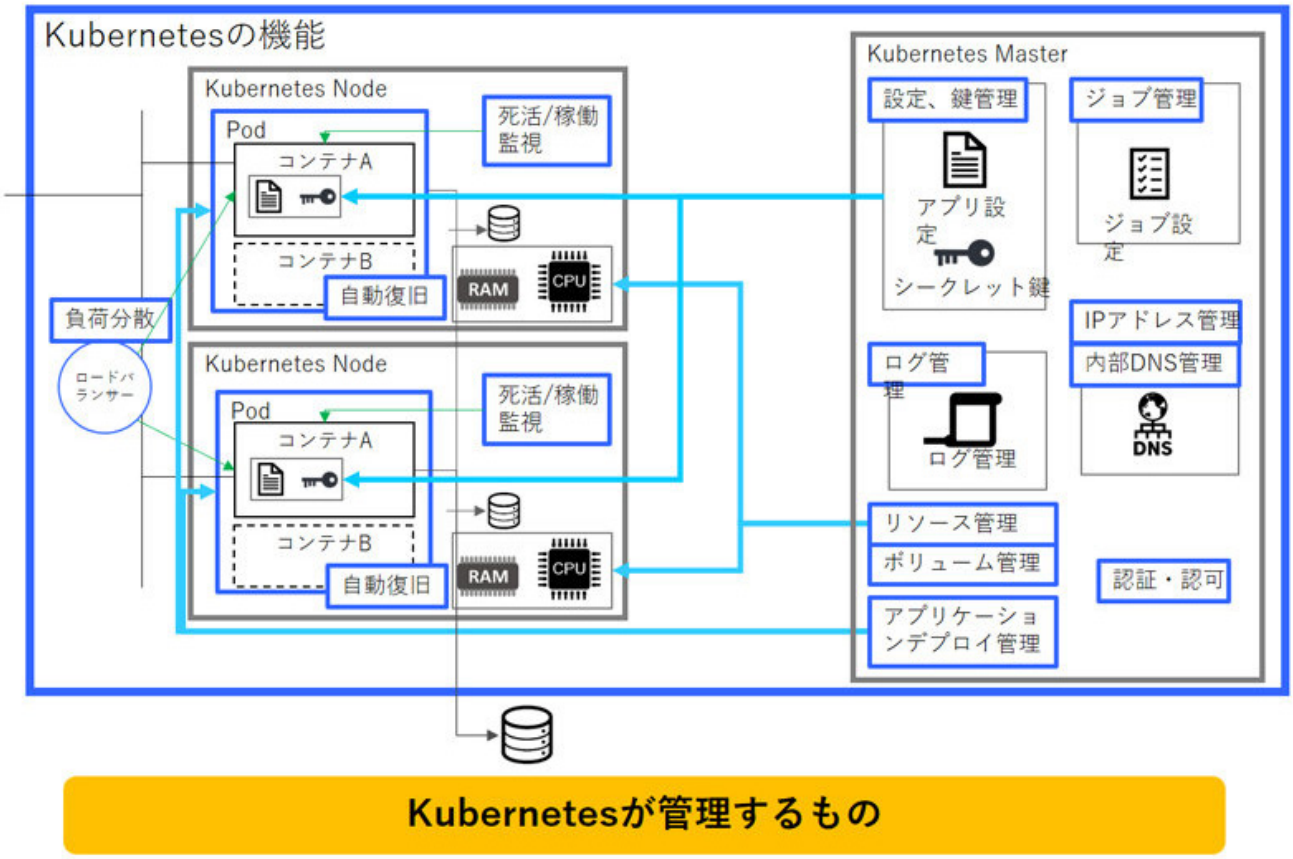
コンテナの実行を「Liveness Probe」と「Readiness Probe」という 2 つの仕組みで監視します。

ロードバランス（可用性向上、負荷分散）

Kubernetes では、コンテナは不変的な存在ではありません。複数のコンテナが起動したり停止したりして状態が遷移します。そのため、それらを束ねるサービスがあります。

アプリケーションの更新

これまでは、アプリケーションの更新は手動で実施していたと思いますが（第 1 回参照）、Kubernetes では「ローリングアップデート」機能や「ブルーグリーンデプロイメント」と呼ばれるクラスタを切り替える手法を利用することで、手動に頼らずアプリケーションの更新が行えるようになりました。



Kubernetes が管理するもの

まとめ

本稿では、Kubernetes が Linux のようなコミュニティーを持っていること、そして、Kubernetes の各種機能と Linux の機能の類似点を紹介しました。

Kubernetes が行っている複数のノードを一元管理する手法は、これまでも「Globus」や「OpenMPI」など HPC（High Performance Computing）での並列計算分野や、仮想基盤を管理する「OpenStack」などで利用されてきました。しかし、複数のノードでアプリケーションをマルチテナントかつマルチタスクで動作させる手法は、Linux のような機構を持つ Kubernetes の登場で初めて実用的に利用できるようになったといえるでしょう。

次回は、Kubernetes を簡単に試す方法と、「どのように Kubernetes を学習していけばいいのか」「どのようなことを理解しなければならないか」を紹介します。

Kubernetes を手元で試せる「Minikube」「MicroK8s」とは

「コンテナ技術」やコンテナ実行環境の「Docker」、大量のコンテナ管理や負荷分散を実現する「Kubernetes」について概要から本番活用の仕方まで解説する「これから始める企業のためのコンテナ実践講座」。第 4 回は、Kubernetes のパッケージマネジャー「Helm」と手元で試せる「Minikube」「MicroK8s」を紹介します。

矢野哲朗，スタイルズ（2019 年 05 月 23 日）

本連載「これから始める企業のための Kubernetes 実践講座」では、第 1 ～ 3 回にわたって、Kubernetes のメリットと主要機能を紹介しました。

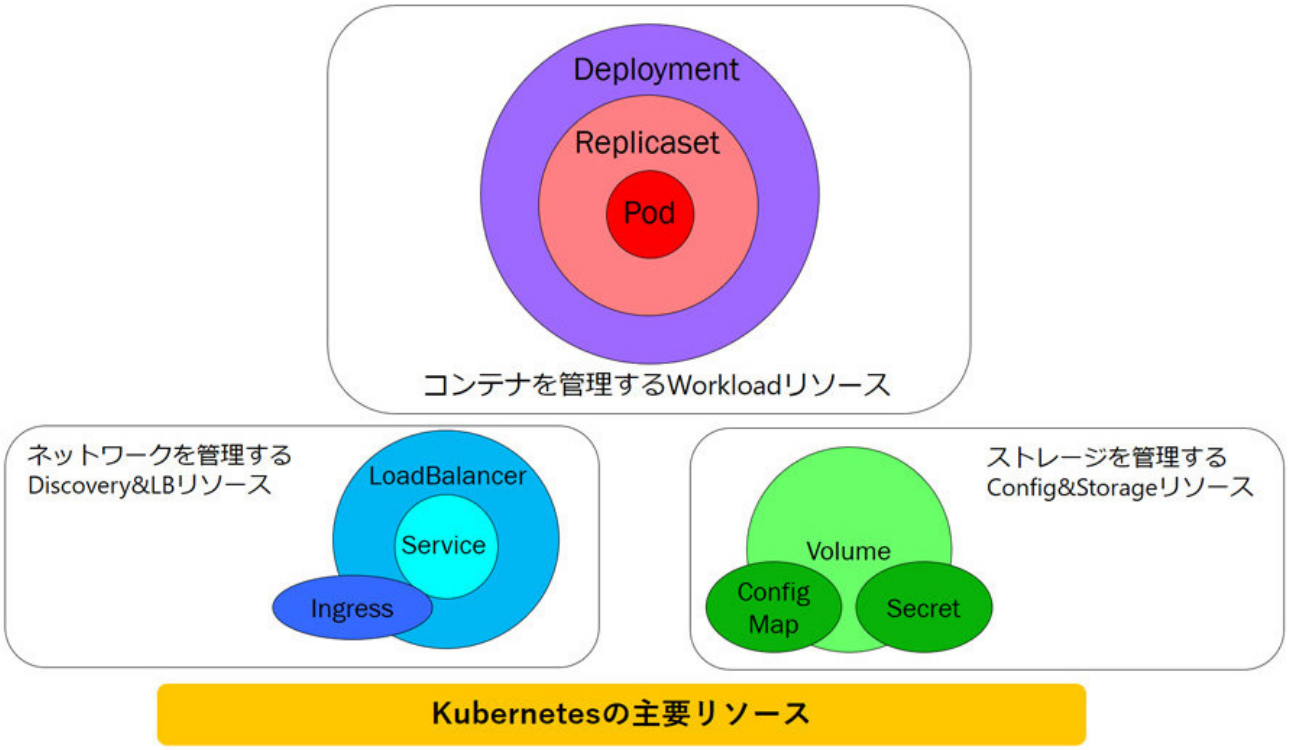
- 「コンテナって何？　どう使える？」――ソフトウェア開発の課題を解決するコンテナ技術（第 1 回）
- 「Excel 手順書にさようなら」――運用管理者の不安を解消する「Kubernetes」のコンセプト（第 2 回）
- Kubernetes がクラウド界の「Linux」と呼ばれる 2 つの理由（第 3 回）

第 4 回は、「リソース」の概念や、リソースを管理するための仕組みを解説します。最後に、Kubernetes をスモールスタートで始める方法を紹介します。

マニフェストファイルのリソースとは

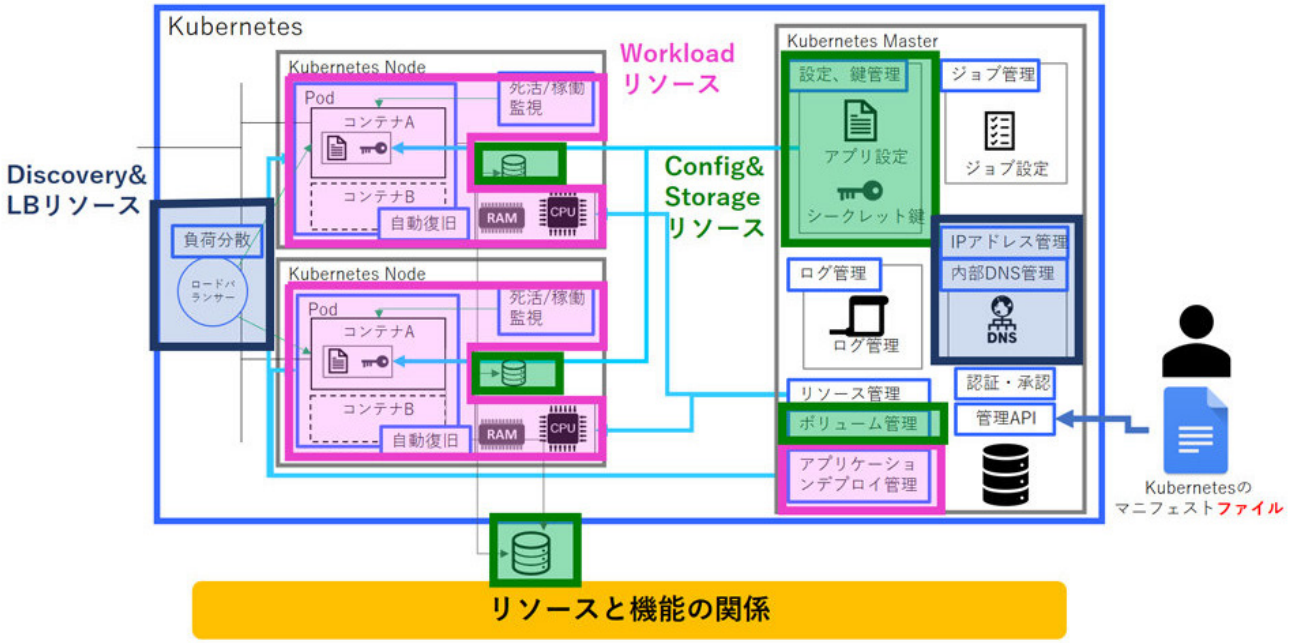
Kubernetes でアプリケーションを運用するには、YAML 形式で書かれた「マニフェストファイル」が必要です（本連載第 2 回、第 3 回参照）。マニフェストファイルでは、コンテナイメージの動作や参照する「リソース」の設定を記述します。代表的なリソースは以下の 5 種類です。

- **Workloads リソース**：コンテナをクラスタで実行するために利用されるリソース
- **Discovery&LB リソース**：コンテナを外部から接続できるようにネットワークを接続したり、コンテナの起動に合わせて負荷分散したりできるようにするためのリソース
- **Config&Storage リソース**：コンテナの設定や機密データ、永続ボリュームを提供するリソース
- **Cluster リソース**：セキュリティやネットワークポリシー、認証など、Kubernetes クラスタを管理するリソース
- **Metadata リソース**：クラスタ内の他のリソースを外から制御できるリソース



Kubernetes の主要リソース

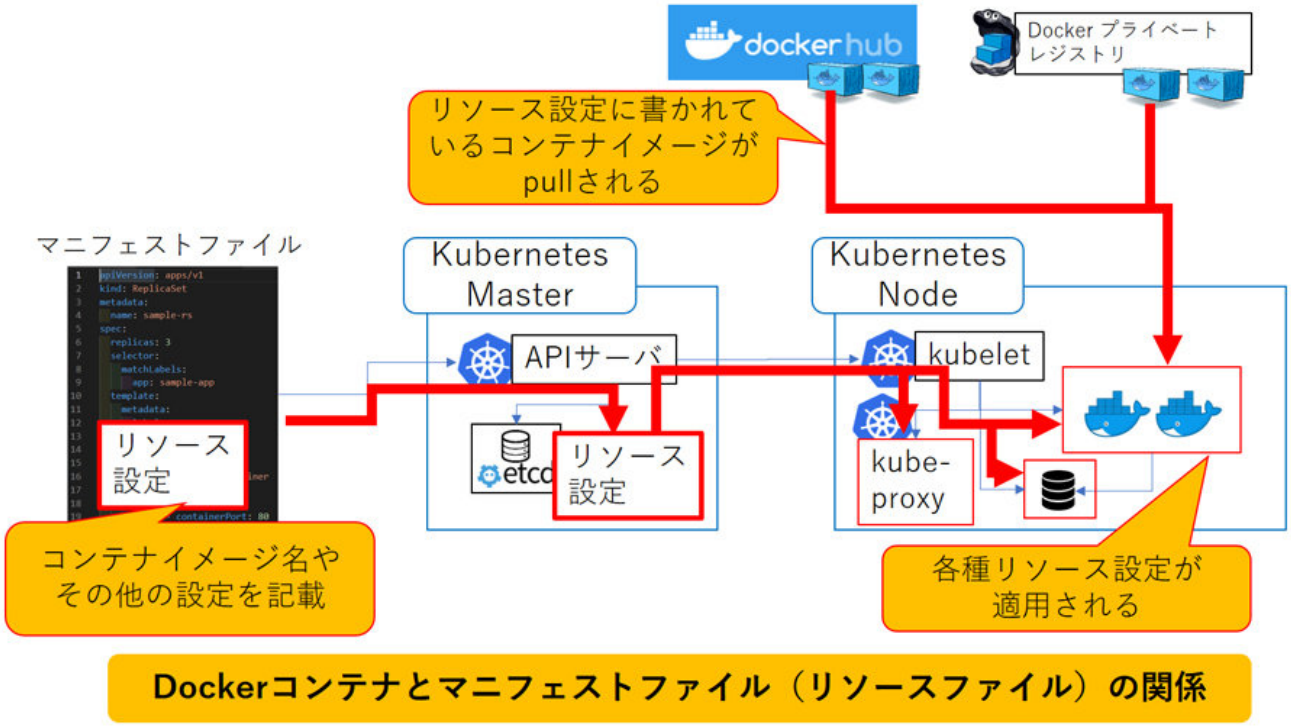
リソースでの設定内容には、コンテナを実行する「Pod」、コンテナの起動数を指定する「レプリカセット」、複数のコンテナをグルーピングする「ラベル」をはじめとして、リソース制限やネットワーク接続、データ保存場所の設定などがあります。



リソースと機能の関係

→目次に戻る

Kubernetes を利用する場合は、マニフェストファイルでリソースを詳細に記述する必要があります。Kubernetes で運用するアプリケーションが増えれば増えるほど、複数のリソース（マニフェストファイル）を管理しなければなりません。アプリケーションが増えた場合は、Docker コンテナイメージを Docker Hub や Docker プライベートレジストリで管理できますが、マニフェストファイルを管理するにはどうすればよいでしょうか。



Docker コンテナとマニフェストファイル（リソースファイル）の関係

現在は、マニフェストファイルの管理を容易にするため、3 種類のプロジェクトが公開されています。

- 「Helm」
- 「kubecfg」
- 「Kapitan」

現在、最も利用されているのは Helm です。本稿では、Helm を紹介します。

Helm とは

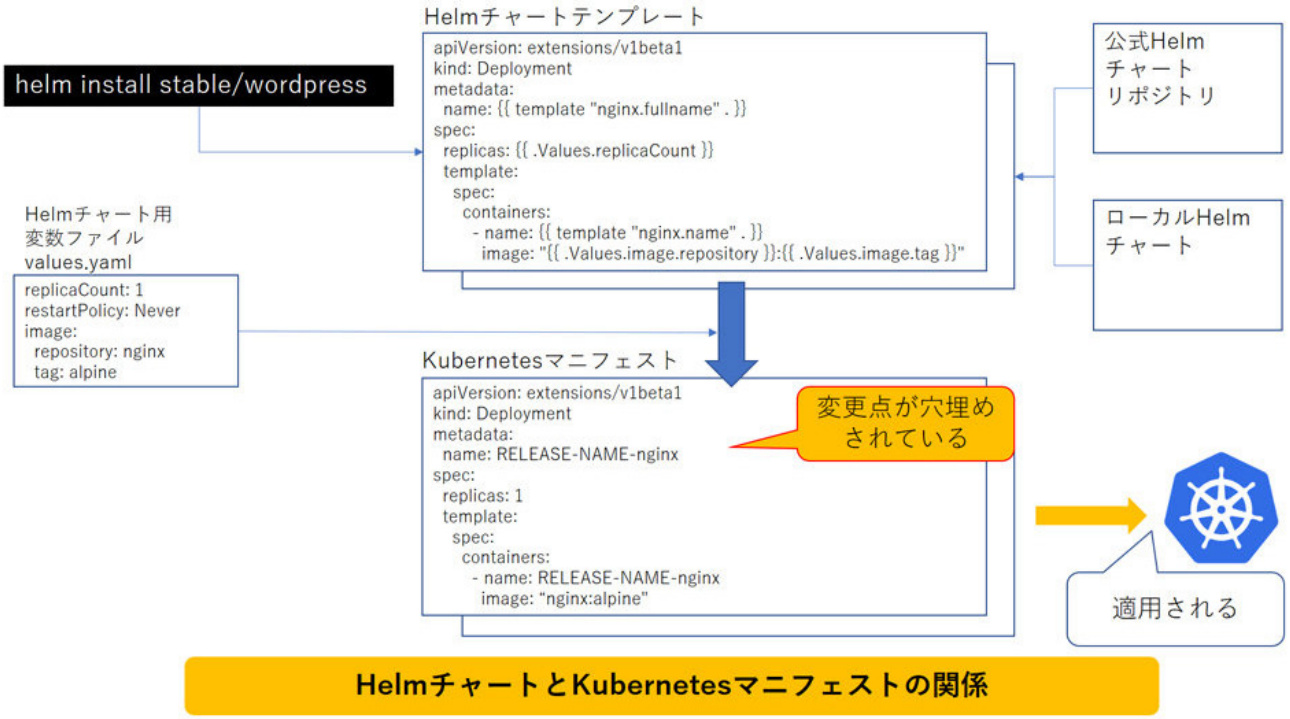
Helm は、マニフェストファイルを「チャート（Chart)」という塊で管理する Kubernetes 用のパッケージマネジャーです。

Helm のチャートは、リソースの設定を丸ごとパッケージングして「Helm Hub」で共有したり、共有されているものを利用したりできる特長があります。

チャートを Kubernetes で利用するには、Kubernetes クラスタで Helm の「Tiller」というコンテナを実行し、Kubernetes クラスタを操作するクライアントに helm コマンドをインストールします。

helm コマンドは、Helm Hub で共有されているチャートを指定してダウンロードしたり、ローカルにあるチャートを指定して Kubernetes にアプリケーションをデプロイしたりすることができます。デプロイする際は引数に変数と値を指定することも可能です。例えば、管理者パスワードを付けてデプロイできます。

Helm を利用する場合は、マニフェストファイルを作成する前に、環境と合致するチャートが用意されているか、Helm Hub で探してみるとよいでしょう。



Helm チャートと Kubernetes マニフェストの関係

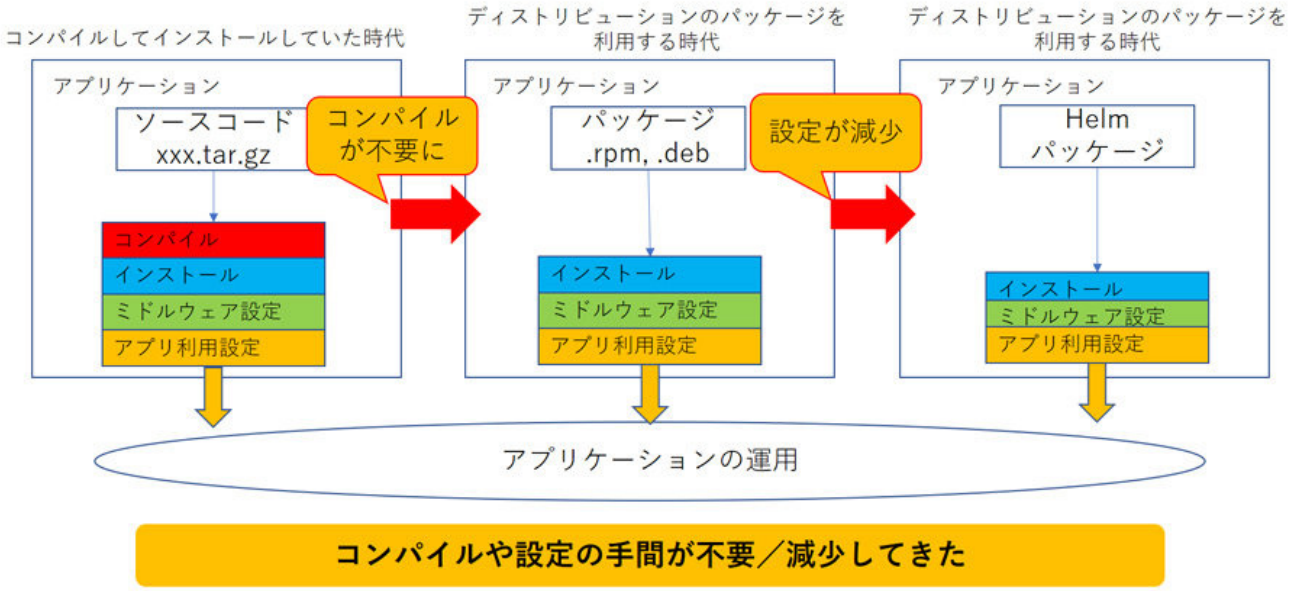
ちなみに、Kubernetes 1.14 では、「Kustomize」という機能が追加されました。Kustomize はパッケージマネジャーに近い機能を提供するコマンドです。マニフェストファイルを別の YAML ファイルに置き換えたり、読み込んだり、編集したりできます。

Kubernetes とパッケージマネジャーによるインフラ管理の変化

Kubernetes のマニフェストファイルを管理するためのパッケージマネジャーが登場し、Kubernetes でより簡単に運用管理ができるようになりました。

Linux でソフトウェアを開発するには、コンパイラとコンパイルの知識が必要でした。それが、rpm や deb などのパッケージマネジャーの登場によってコンパイルの知識が不要になり、アプリケーションの設定に関する知識だけあればよくなった状況に似ています。

今後は、「アプリケーションをインストールして設定して安定稼働させる」というインフラエンジニアの仕事はますます、「Kubernetes でアプリケーションを実行できるように環境を整備する」というものに置き換わっていくかもしれません。



コンパイルや設定の手間が不要／減少してきた

→目次に戻る

Kubernetes の利用ケース

Kubernetes はどのような場面で利用するのに適しているでしょうか。最も利用しやすいのは、自社で提供しているコンシューマー向けサービスの運用です。

Kubernetes を本番環境で運用するには、開発者と運用者の連携が必要不可欠です。開発者が自社にいるようなコンシューマー向けサービスの場合、社内で密に連携を取れるため、適用しやすいといえます。では、社内で利用している業務システムではどうでしょうか。

まず、業務システムをオープンソースソフトウェア（OSS）で構築している場合は、Kubernetes で運用できるでしょう。その理由は、Docker が提供する Docker コンテナイメージ共有サービス「Docker Hub」で、OSS の Docker コンテナイメージが提供されている可能性が高いからです。

Docker Hub で提供されている Docker コンテナイメージを利用して、自分でイメージを作成することなく環境を構築してアプリケーションをデプロイして、Kubernetes で運用できるというわけです。

一方、OSS を使わずに開発したシステムや、ベンダーから購入したアプリケーションパッケージを Kubernetes で運用するのは難しいかもしれません。Docker イメージで構築していない場合は、コンテナイメージを作成する必要があるからです。購入したアプリケーションパッケージで業務システムを構築している場合は、アプリケーションパッケージを Docker イメージに組み込むことを販売元のベンダーが認めているか、ライセンスを確認したり、問い合わせたりする必要があるでしょう。

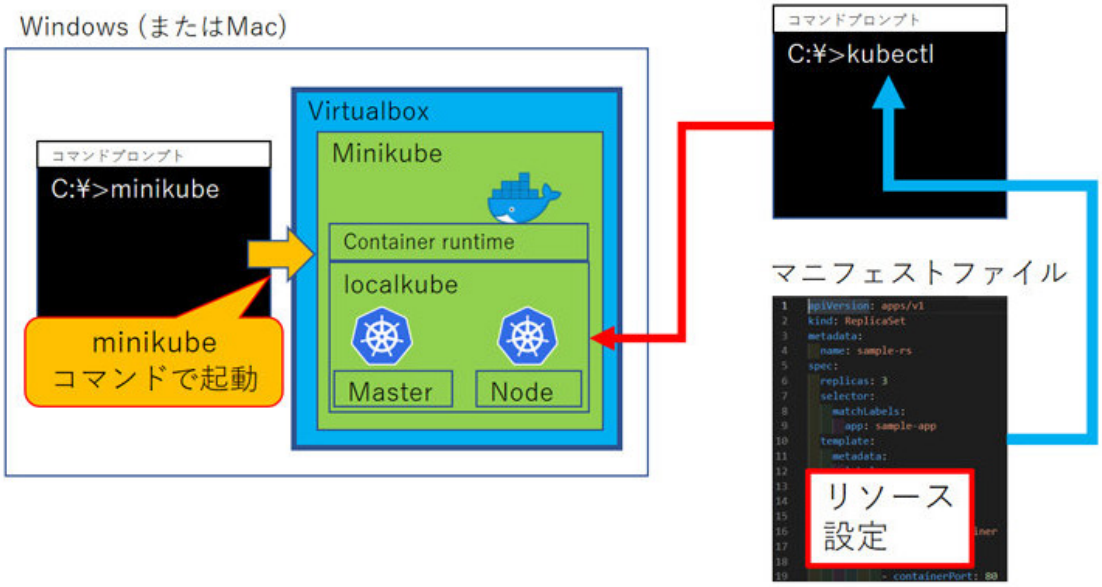
本連載第 1 ～ 3 回でお伝えしてきたように Kubernetes が運用のハードルを下げることは間違いありませんが、Kubernetes は仮想マシンを社内で稼働させるものではないため、「全社内システムを簡単に Kubernetes で運用できる」というわけではないことを覚えておくとよいでしょう。

Kubernetes を手を使って学んでみるには？

Kubernetes のマニフェストファイルを管理できるパッケージマネジャーや Kubernetes を利用できるケースを解説しました。それでは手を使って Kubernetes を学んでいきましょう。まず、手元の環境で Kubernetes を始める方法を紹介します。

1 つ目は、「Docker for Windows」「Docker for Mac」をインストールして Kubernetes を実行する方法です。Windows OS の場合は、「Windows 10 Pro」以上で、仮想化機能（「Intel VT」か「AMD-V」）を有効化して「Hyper-V」が動作するようにしておく必要があります。

2 つ目は、Windows または Mac に「Oracle VM VirtualBox」をインストールして「Minikube」を実行させるという方法です。こちらも仮想化機能を有効化している必要があります。



Minikube利用構成

Minikube 利用構成

しかし、この 2 つは Windows ／ Mac 環境に仮想化した OS を立ち上げて Kubernetes を実行するため、ネットワーク環境がトリッキーになるという一面があります。「やってみたけどうまく実行できない」「取りあえず実行できたが、Kubernetes につながらない」というトラブルに見舞われるかもしれません。

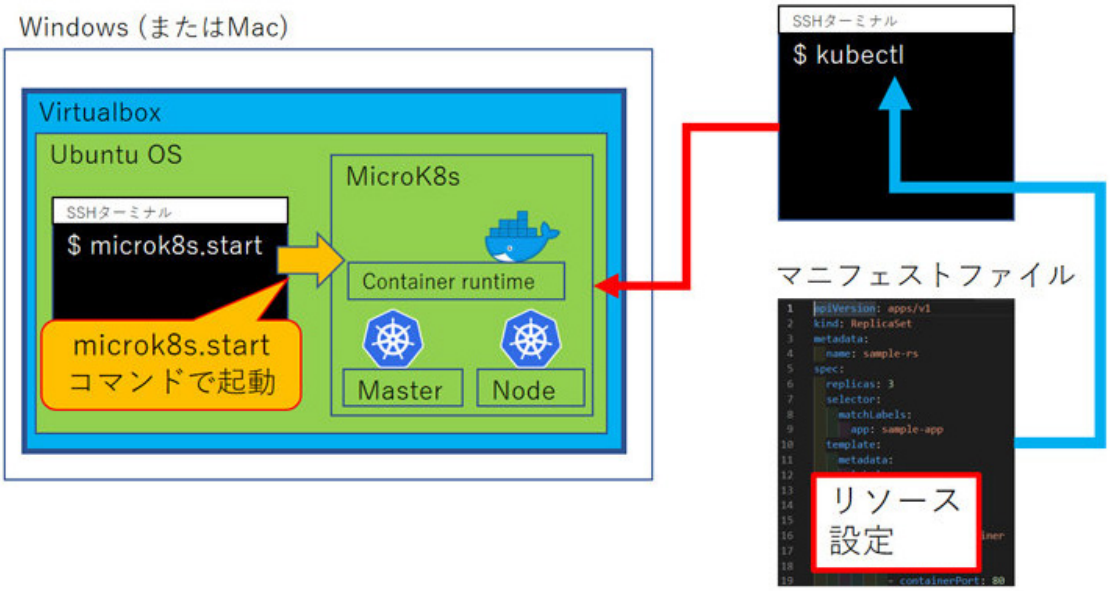
そこで、お勧めするのが、3 つ目の「MicroK8s」を用いて Kubernetes 環境を構築する方法です。

MicroK8s による Kubernetes 環境の作成

MicroK8s は、Ubuntu が開発している Kubernetes ディストリビューションの一つで、以下の特長があります。

- VirtualBox がなくても動く（＝仮想サーバでも動かせる）
- 1 ホストで軽量に動く（約 1 分で起動）
- 1 コマンドでインストールやバージョンアップ、リセットができる
- 「Dashboard」「Docker Container Registry」「GPU Passthrough」「Istio」が簡単に有効化／無効化できる
- CNCF（Cloud Native Computing Foundation）から Kubernetes ディストリビューションとして認定されている
- ネットワーク設定に関するトラブルを避けられる
- Docker レジストリが組み込まれているので、自分でビルドしたコンテナイメージを外部にアップロードせず保存できる

Microk8s は、1 ホストで実行するので、通常の Kubernetes クラスタのように複数台で連携する機能はありません。それ以外は Kubernetes と同じ機能を持ちます。



MicroK8s利用構成

MicroK8s 利用構成

Linux ディストリビューションが実行できる PC を 1 台用意して、Ubuntu をインストールする必要はありますが、原因が分かりにくいネットワークの問題を避けられるメリットがあります。

もちろん、サーバで仮想化された独立した Linux サーバを作成でき、そこに Ubuntu をインストールできれば、それでも構いません。独立した Linux サーバまたは、仮想サーバに Linux サーバを立ち上げて Minikube を実行する方法もあります。

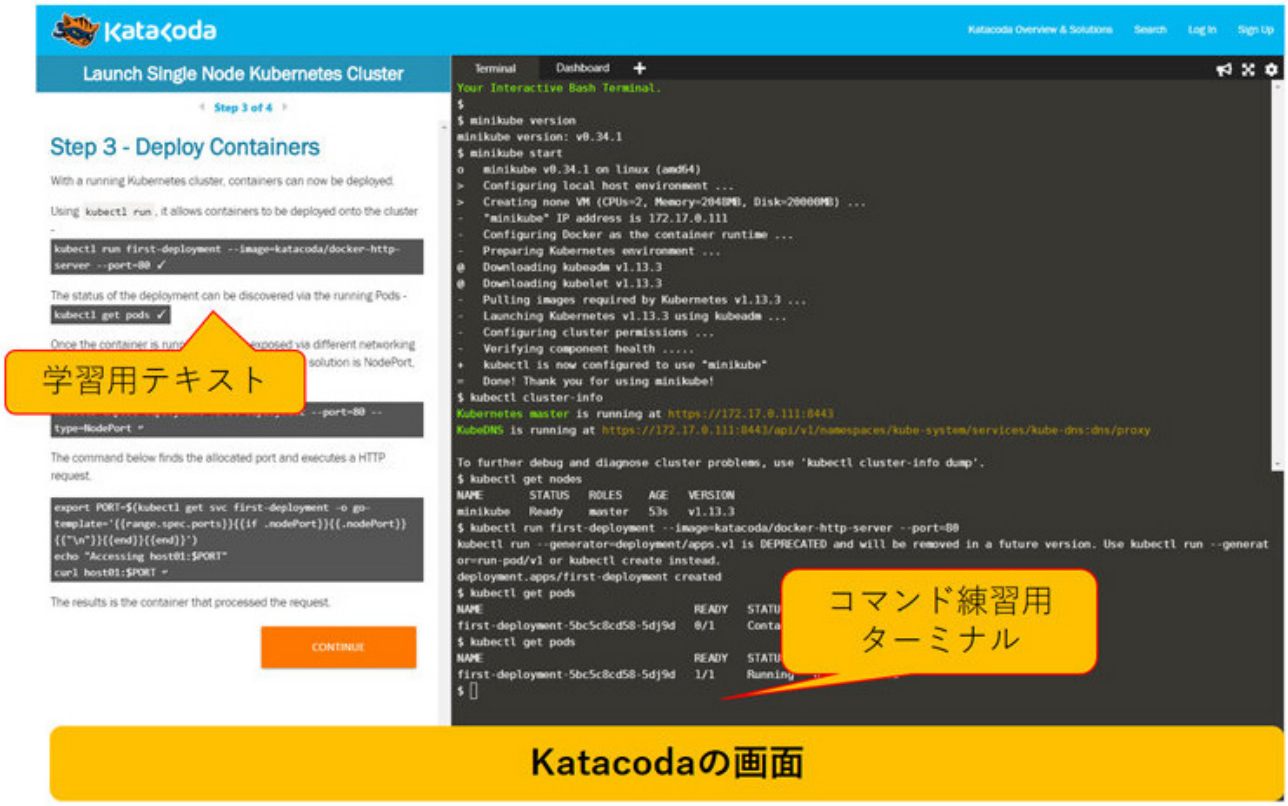
しかし、Minikube は VirtualBox を利用するので、仮想サーバの Linux に VirtualBox で仮想マシンを実行することになります。もし、仮想サーバで仮想マシンを実行させる機能が有効になっていなければ、実行できません。そういう場合には、MicroK8s の利用がお勧めです。

そもそも「Linux サーバを準備できない」という場合は、「Katacoda」で学ぶ方法があります。

Katacoda とは

Katacoda は、ブラウザから無料で勉強用のインスタンスを起動できる Web サービスです。インスタンスの起動後、表示されるテキストに沿って操作を学べます。

左側に説明、右側にコンソールの画面があり、左側を見ながら右側のコンソールでコマンドを入力して操作方法を理解します。テキストは英語ですが、さほど長い文章ではないので、Google 翻訳などで読みながら、学習を進められると思います。操作時間に制限はありますが、どういう雰囲気で動かすかを理解するには最適な教材です。



Katakoda の画面

Kubernetes のリソースという概念、そしてリソースを管理するファイルを管理する方法、スモールスタートで始めるためにどのような方法があるかを紹介しました。

最近は、Kubernetes に関する日本語の書籍が充実してきており、学ぶハードルも下がってきています。これらのツールなどを通じて、今後インフラエンジニアの仕事を変えるかもしれない Kubernetes の機能や構成管理などを理解していきましょう。

次回は本連載の最終回として、Kubernetes を本番で実行する環境を構築する方法と、便利な周辺ツールを紹介します。

マネージドサービス？ 自前構築？ Kubernetes を本番環境で使うにはどうすればよいのか

大量のコンテナ管理や負荷分散を実現する「Kubernetes」について概要から本番活用の仕方まで解説する本連載。最終回は、Kubernetes はどのような環境で構築すればいいのかについて、具体的な方法やポイントを解説します。

矢野哲朗，スタイルズ（2019 年 07 月 01 日）

本連載「これから始める企業のための Kubernetes 実践講座」では、Kubernetes のメリットと主要機能を紹介してきました。最終回となる今回は、Kubernetes はどのような環境で構築すればいいのかについて、具体的な方法やポイントを解説します。

Kubernetes の 4 つの構成方法

Kubernetes の構成方法には、ニーズや環境、役割によって 4 つのパターンが考えられます。

- クラウド（以下、パブリッククラウドのことを指す）事業者のマネージドサービスを利用する
- クラウドの IaaS 上に自前で構築する
- オンプレミスのサーバ環境に構築する
- オンプレミスの開発端末に構築する

	マネージドサービス	自前構築
クラウド	クラスタ構成	クラスタ構成
オンプレミス		クラスタ構成
		単体構成

Kubernetes構築環境パターン

この中で一番簡単な方法は、クラウド事業者のマネージドサービスを利用することです。Amazon Web Services（AWS）の EKS（Amazon Elastic Container Service for Kubernetes）、Google Cloud Platform（GCP）の GKE（Google Kubernetes Engine）、IBM Cloud の IKS（IBM Cloud Kubernetes Service）、Microsoft Azure の AKS（Azure Kubernetes Service）などがあります。

これらのマネージドサービスを利用するメリットは、サーバインフラを持たなくてよいということのみならず、サーバの管理運用についてもクラウド事業者にお任せできるところにあります。Kubernetes の管理（バージョンアップや死活監視）についても工数を削減できるので、Kubernetes の専任技術者を置けない会社向けといえるでしょう。

また、クラウド事業者が提供する IaaS を利用し、自前で構築、管理、運用するパターンもあります。マネージドサービスに比べると楽ではありませんが、既存のシステムがクラウドで稼働している場合にはさまざまツールやサービスと組み合わせることで、Kubernetes のみならずインフラ全体として運用を軽減することにつながるでしょう。

クラウドを会社のポリシーで使えない場合や、ワークロードがマッチしない場合もあるでしょう。そういう場合はオンプレミスで構築し、管理することになります。クラウドサービスの恩恵は受けられませんが、既存リソースを流用できるといったメリットもあります。

また前回紹介した「Minikube」「MicroK8s」を使って、手元の開発端末で動作チェックのために動かす場合もオンプレミスでの構成と同様になります。

さらに、本番環境はクラウドのマネージドサービスで動かすが、開発環境や検証環境はオンプレミスの開発端末やサーバで動かすといった用途では、環境を 2 つ作って構成することも考えられます。

そこで、ここからはマネージドサービスとオンプレミスの両方で共通に理解しておかなければならないことと、それぞれ特有のことを解説します。

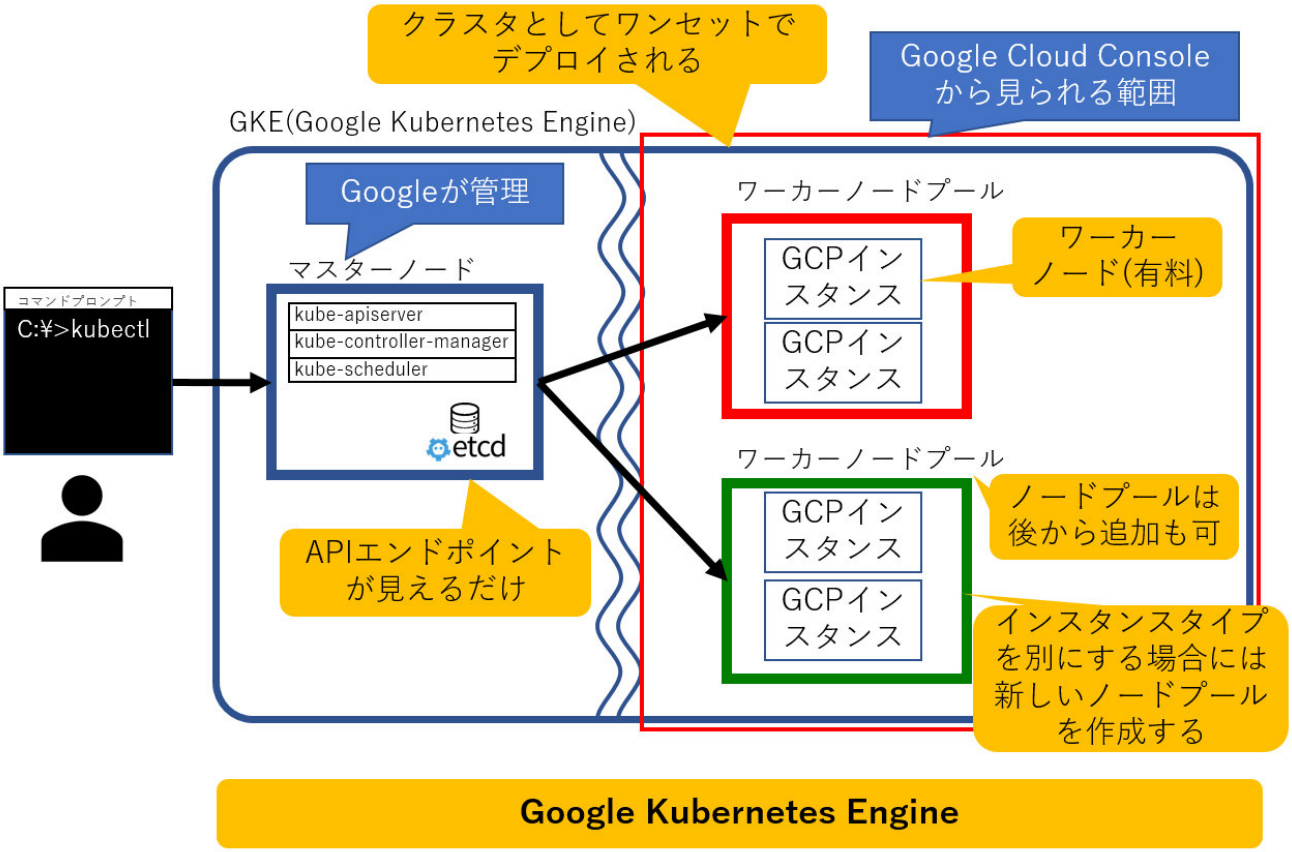
Kubernetes におけるマスターノードとワーカーノード

連載第 3 回でも書きましたが、Kubernetes には「コントロールプレーン」（マスターノード）と「データプレーン」（ワーカーノード）というものがあります。このコントロールプレーンとデータプレーンの見え方がマネージドサービスとオンプレミスで異なりますし、費用がクラウドごとに異なるので注意が必要です。

マネージドサービスの場合

マネージドサービスでは、マスターノードを各クラウド事業者が管理します。マスターノードは利用者からは見えないので、再起動もできません。Kubernetes の管理 API エンドポイントが提供されているだけです。

例えば GCP の GKE では下図のように、マスターノードを見ることはできません。マスターノードに費用はかからず、費用がかかるのはワーカーノードのみとなっています。



オンプレミスの場合

一方、オンプレミスでの Kubernetes サーバを構築する場合、マスターノードとワーカーノードの両方を管理、運用しなければなりません。そのため、Kubernetes の管理に関する知識がある程度必要になります。

オンプレミスで困ることとして「マスターノードの構築と運用管理」「ストレージの管理」「ネットワークのロードバランサーの管理」の 3 つが挙げられます。

- **マスターノードの構築と運用管理**

Kubernetes が出始めたころ、オンプレミスで、ツールを使わずにマスターノードを構築するのは、「苦行」といわれるほど大変な作業でした。しかし現在では、CNCF 提供の Kubernetes インストールツールをはじめ、kubeadm、kubespray、kops、rke など、作業を楽にしてくれるツールがたくさん出てきています。それらを使うことにより気軽にマスターノードを構築できるようになりました。

Certified Kubernetes - Installer

さらに Linux ディストリビューションを出している会社や独立系の開発会社がプラットフォームとして構築できる Kubernetes ディストリビューションを有償サポート付きで提供しています。

これらのさまざまなツールやディストリビューションがありますが、どれも Kubernetes のソースコードを利用しています。これは各種 Linux ディストリビューションが Linux カーネルのソースコードを利用しているのと同じです。

「Kubernetes ディストリビューションは、kubeadm で構築された Kubernetes 環境とどう違うのか？」と聞かれることもあります。これは「各 Linux ディストリビューションが Linux カーネルとどう違うのか？」という質問と似ています。この質問に対する回答は、「Linux ディストリビューションは、Linux カーネルとは違うところもあるが、Linux の挙動という意味ではどちらも同じ機能を有している」です。

Kubernetes はさまざまなニーズと環境に合わせることのできる拡張性を持ったプロダクトです。この柔軟性により、さまざまなバリエーションのディストリビューションが生まれています。基本的な挙動については、CNCF から認証を受けていれば同じといえるでしょう。拡張機能により追加された部分や機能については、ディストリビューションにより異なるところがあるでしょう。

運用管理という意味では、Kubernetes を自前で使う場合、サポートリリースサイクルに注意する必要があります。正式なサポートはマイナーリリース 3 つ分までメンテナンスされます。マイナーリリースは 3 カ月ごとなので、正式なサポートは 9 カ月ということになります。Linux ディストリビューションから出ている Kubernetes については、サポートなどのサブスクリプション契約によりサポート期間が異なります。

サポート期間の長い短いはありますが同じバージョンをずっと使い続けるのは避けましょう。新しいバージョンでは機能がどんどん追加され使いやすくなっていくことを思えば、「Kubernetes 自体のアップグレード戦略をどうするか?」についても考えておいた方がいいでしょう。各社のディストリビューションではアップグレードするツールを提供しています。

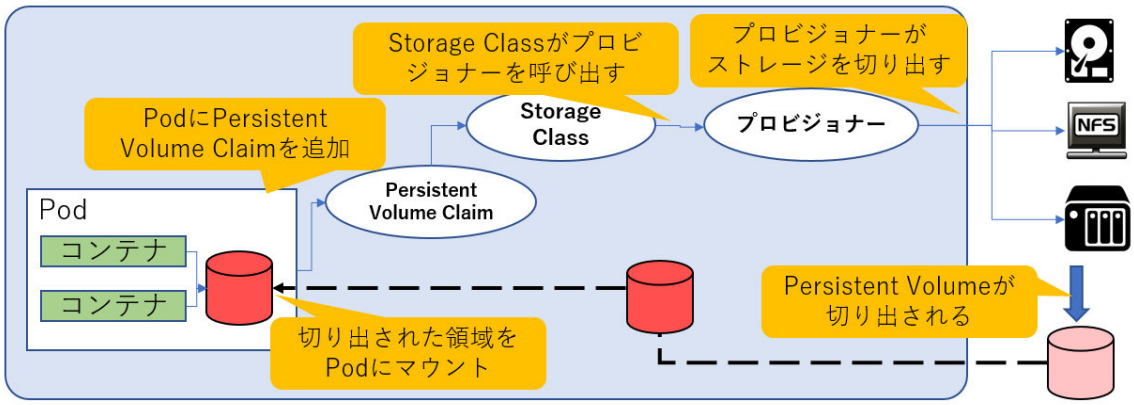
・ストレージ（Persistent Volume）の管理

コンテナ内のデータはコンテナの停止とともに消滅してしまいます。そのままだと保存しなければいけないデータについても消えてしまいます。それを避けるために、Kubernetes は「Persistent Volume」という仕組みを備えています。これは Pod 内にコンテナ用のデータ保存領域を提供してデータの消滅を防ぐものです。Kubernetes には、この Persistent Volume に必要な領域を動的に切り出すための「ダイナミックプロビジョニング」という機能があります。

一般的にはストレージといっても、さまざまなものがあります。サーバにつながっている HDD から、ネットワーク経由の iSCSI や分散ファイルシステム、仮想マシンの仮想ディスクなど、種類やスピード、接続方法まで多種多様です。

Kubernetes は、それらを一元的に扱おうとします。しかし、多種多様なストレージの、接続方法までは管理できないため、「プロビジョナー（Provisioner）」という仕組みでそれらの違いを吸収します。プロビジョナーはそれらのストレージを動かすためのデバイスドライバのような働きをします。

デバイスドライバで違いを吸収してもらいたい一方で、さまざまなオプションは個別に指定したいこともあるでしょう。それらのオプションを指定するのが「ストレージクラス（Storage Class）」です。名前を任意に指定できるので、別々の環境（例えば、GCP の GKE と AWS の EKS）で、名前だけ同じにしたストレージクラスを用意してマニフェストファイルで一意に利用することも可能です。通常はデフォルトのストレージクラスが用意され、指定がなければそれが使用されます。



Storage Classを使った動的プロビジョニング

→目次に戻る

プロビジョナーはさまざまなストレージに対応していますが、それらにマッチしたストレージを保有していなければなりません。マネージドサービスの Kubernetes であればクラウド事業者が用意しているブロックデバイスサービス（AWS の EBS など）を利用すればよいのですが、オンプレミス環境ではそういったブロックデバイスを切り出せるストレージを用意しなければなりません。

ストレージ各社もこぞって対応を始めていますが、現状で一番お手軽に利用できるのは NFS です。NFS 利用時に注意しなければならない点としては、デバイスへの書き込み遅延に厳しいアプリケーションや、データを激しく書き換えるアプリケーションでは、パフォーマンスの面から問題が生じる可能性が高いということです。こういったアプリケーションでは利用しないようにしましょう。なお NetApp のストレージを既存リソースとして利用できる場合は、ONTAP のバージョン 8.3 以降であれば「Trident」というプロビジョナーを使うのもいいと思います。

- **ネットワークのロードバランサー（Service リソースの type：Loadbalancer）の管理**

オンプレミスで問題となるのが、Kubernetes の外のネットワークから Kubernetes 内に接続する方法です。マネージドサービスでは、Service リソースの type：Loadbalancer により、クラウド上のロードバランサーサービスを利用して外部ネットワークとの接続が自動的に設定されるようになっています。しかし、オンプレミスではそうなっていないので、「MetalLB」などのツールを使うことになるでしょう。

コラム「Kubernetes の管理 UI」

Kubernetes は、基本的にマニフェストファイルであったりコマンドであったり、CI/CD などのパイプライン自動化ツールと一緒に使いやすくなっています。しかし、利用する人の全てがそれらのツールに慣れていたり、知識を十分に持っていたりするわけではありません。ブラウザからグラフィカルに閲覧し、設定を変更したい場合もあるでしょう。

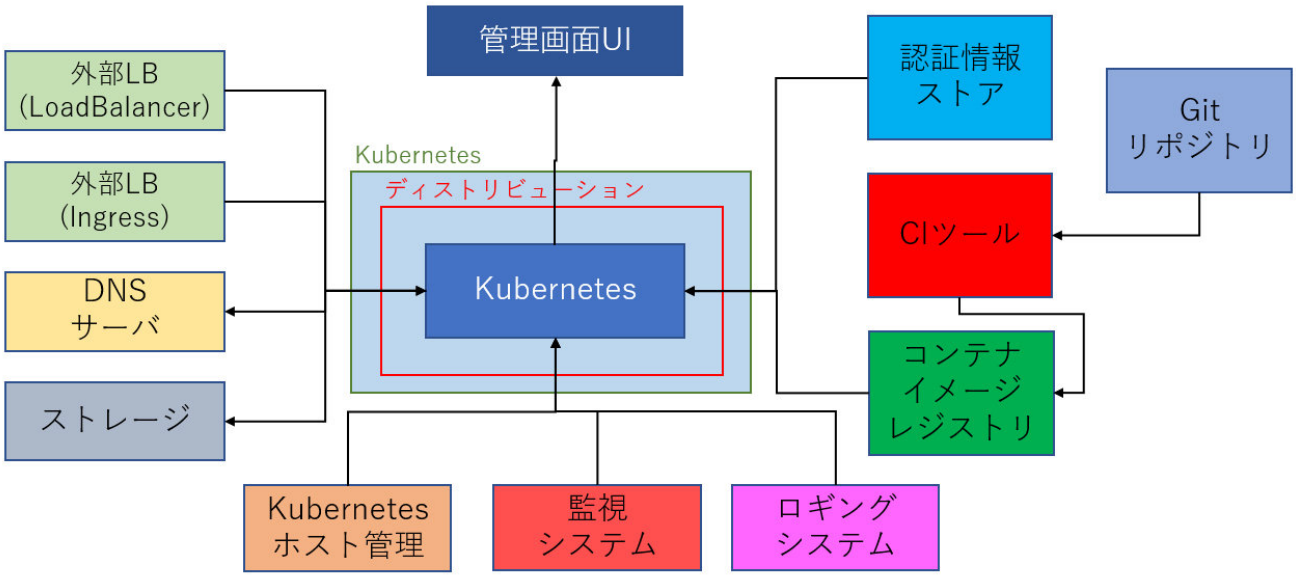
Kubernetes のダッシュボードは必要な基本情報を見るためには十分な機能を持っています。さらに利便性を向上させブラウザでの UI を提供してくれるソフトウェアとして Rancher Labs の「Rancher」や Weaveworks が提供するソフトウェアがあります。

Kubernetes とうまく付き合っていこう

本連載では、「Kubernetes を使うと、何が変わるのか？」「Kubernetes を使うには、何が必要なのか？」といったことをインフラ技術者の視点から見てきました。

インフラ技術者として考えなければならないことは、Kubernetes 自体の挙動と管理方法の理解、Kubernetes を有効に動かすための周辺アプリケーションの設定です。便利に使えるようになるためには、苦労しなければいけないことが非常にたくさんあるように見えるでしょう。

最終回で紹介したこと以外にも、対応しなければならない課題があります。認証情報サーバとの接続方法やログを取得して保存する方法、監視方法、バックアップ方法、「サーバの負荷監視をどうするか」「コンテナイメージレジストリをどうするか」「DNS のレコードを動的に書き換えるにはどうすればよいか」などです。これに CI/CD 環境も考慮すると検討課題はさらに増えていきます。



Kubernetesと周辺システム

こういったさまざまな課題を軽減できるので、Kubernetes を使うには、マネージドサービスでの利用を検討してはいかがでしょうか。

一方で、オンプレミスで Kubernetes を利用しなければならない場合もあるでしょう。ここまで紹介した全ての課題を一度に解決しようとせず、一つずつ解決していくことをお勧めします。本稿で紹介した各ディストリビューションや有償サポートをうまく利用することも有効でしょう。課題に対応したツールやソフトウェア、コンサルティングを利用するなど、Kubernetes とうまく付き合っていけるようになることを願っております。

筆者紹介

矢野 哲朗

株式会社スタイルズ SI ビジネスグループ シニアエキスパート

ネットワークから DB とストレージ、パフォーマンスチューニングに従事。「ownCloud」と「Nextcloud」からオープンソースへの貢献を深め、現在はコンテナに関する企画、SI に従事している。

棚田 美寿希

株式会社スタイルズ マーケティング・広報担当

社内初のコーポレート広報担当として、企画、計画〜運営に従事する。自社サービスや取り扱うオープンソースのマーケティング、PR も兼任。非エンジニアならではの視点で、技術情報や IT 事例を世に広めていけるよう日々まい進している。

編集：@IT 編集部
発行：アイティメディア株式会社