# コンテナ用語集

コンテナ開発とそれに関連するテクノロジーから数多くの用語が生まれていますが、この記事ではそれらの用語の意味について説明します。

1つのテクノロジーに対して数多くの専門用語が存在することがありますが、コンテナはその最たる例です。ソフトウェア仮想化の1つの形態であるコンテナは、仮想マシンよりはるかにスケーラブルで比較的作成が容易なうえ、ツールとサポートソフトウェアの巨大なエコシステムが存在し、サーバーソフトウェア開発の新しいデフォルトアーキテクチャーとなっています。

この記事では、コンテナとコンテナ関連の用語について説明します。大部分の用語はコンテナの開発や運用に直接関連するものですが、その多くは、一般的にコンテナに関連して使用されるだけでなく、コンテナと関係のない目的でも使用されるという点でコンテナと近い関係にあります。

## アジャイルソフトウェア開発

従来のトップダウン型のプロジェクト管理に代わる、反復型ソフトウェア開発で使用される一連の手法。アジャイル開発では、コラボレーションとチームによる意思決定に重点が置かれます。アジャイルの12の主要な概念は、アジャイルマニフェストで説明されており、これには実用的なソフトウェアを (週単位や月単位で) 頻繁に提供したり、実用的なソフトウェアを進捗確認の主な手段とみなしたり、最も効果的なコミュニケーションの方法として対面で会話したり、最高のソフトウェア製品は自己管理型チームの成果であると考えたりといった概念が含まれます。アジャイルにおいては、うまくいっていること (またはいないこと) とチームの効率を向上させる方法を定期的に分析することがきわめて重要です。

## アプリケーションプログラミングインターフェイス (API)

アプリケーションを構築して接続するための一連のルーチン、プロトコル、ツール、およびルール。オペレーティングシステム、アプリケーション、またはWebサイト用のさまざまな種類のAPIがあります。APIは、さまざまな人が作成したソフトウェアで構成され、さまざまなコンピューターやネットワーク上で動作するソリューションを開発者が作成しやすくするルールを規定して適用します。

## ベアメタル (ベアメタルサーバー)

物理マシン。この用語は、一般的に物理サーバーと仮想マシンやクラウドでホストされるサーバーレスアプリケーションを区別するために使用されます。

## ブルー/グリーンデプロイ

ダウンタイムを最小化し、必要に応じて迅速にロールバックを行う継続的デプロイをスムーズに進めるための展開戦略。それぞれが別バージョンのアプリケーションを実行する2つの同じ環境 (現在の本番バージョン向けのブルーと新しいバージョン向けのグリーン) があり、ユーザートラフィックはすべてブルーにルーティングされます。そして新しいバージョンが最終テストに合格すると、すべての着信要求をグリーン環境に送信するようにルーターが再構成される一方、ブルーはアイドル状態になります。これは、A/Bまたはレッド/ブラックデプロイと呼ばれることもあります。

## カナリアデプロイ

ブルー/グリーンおよびローリングデプロイ戦略を組み合わせたもの。新しいコードを本番環境のごく一部に展開し、そのような新しい展開に少数のユーザーをルーティングすることで潜在的な影響を最小限に抑えます。エラーが報告されなければ、新しいバージョンを本番環境の残りの部分に段階的に展開できます。この用語は、カナリアに有毒ガスを警告させる炭鉱作業員の慣習を引用したものです。

## Kubernetes適合性認定プログラム (CNCP)

Kubernetesのすべての認定バージョンが必要なAPIをサポートしていることを確認する、Cloud Native Computing Foundationが実施する適合性テスト。Kubernetesをベースとするソフトウェアを提供する企業はすべて、自社製品に対する一連のテストを実施し、その結果をCNCFに提出できます。認定を維持するには、少なくとも1年に1回はKubernetesの新しいバージョンのすべてでソフトウェアをテストする必要があります。テストアプリケーションであるSonobuoyはオープンソースであり、ディストリビューションの適合性を確認するために誰でも実行することが可能です。

## クラウドネイティブ

分散アプリケーションを示す用語であり、それらを構築して実行し、クラウドコンピューティングモデルを活用して柔軟な拡張と優れた可用性を実現するアプローチ。クラウドネイティブアプリケーションは、一般的にはオープンソースであるかオープンソースのコンポーネントをベースとしており、それらを動的にオーケストレーションしてリソース稼働率を最適化できるコンテナでマイクロサービスとして実行されます。

## コンテナ

さまざまなコンピューティング環境でアプリケーションを迅速かつ確実に実行できるよう、コードとその依存関係のすべてをパッケージ化するソフトウェアの単位。コンテナイメージは、アプリケーションコード、ランタイム、システムツール、システムライブラ

リ、および設定を含む実行ソフトウェアです。コンテナは、オペレーティングシステムやハードウェアではなくアプリケーションのみを仮想化するため、仮想マシンとは異なります。クラウドの拡張性と信頼性を得るためにコンテナでマイクロサービスとしてアプリケーションを実行するのが一般的なアプローチです。

# コンテナオーケストレーター

通常は自動化された方法で一連のコンテナの監視と管理を行うためのツール。現在のところ、Kubernetesが主要なコンテナオーケストレーターとなっています。

# 継続的インテグレーション/継続的デリバリ (CI/CD)

自動化を活用してお客様に頻繁にアプリケーションを提供する手法。CI/CDプロセス全体はパイプラインと呼ばれることがあり、パイプラインを構成する自動化されたツールはツールチェーンと呼ばれます。
次の3個の用語は、CI/CDに関連するものです。

**継続的インテグレーション**
開発者が (通常は1日に複数回) 共有レポジトリにコードを統合する必要がある手法。チェックインしたコードのすべてのユニットが自動化されたビルドとテストで検証されるため、チームは早い段階で問題を検出し、アプリケーションコンポーネントを短期間で出荷できます。

**継続的デリバリ**
開発、テスト、本稼働へとコードを移行するために使用する自動化された手法。自動化により、チームはアプリケーションデリバリを遅らせる可能性がある手動のプロセスの負担から解放されます。

**サーバーレスコンピューティング**
バックエンドのサービスが従量制で提供され、開発者が基盤となるインフラストラクチャを気にせずにコードを作成して展開できるクラウドコンピューティングモデル。サーバーレスアプリケーションは、通常オンデマンドで提供され、自動的に拡張されるため、企業は使用した分の料金を支払うだけで済みます。

# 分離

分ける、または離すこと。分離されたコンポーネントは、さらなるアウトプットを生み出すために連携するときも自律的に動作し、お互いを認識しません。たとえば、パブリッククラウド環境では (プロバイダーが管理する) 物理インフラストラクチャが (IT部門やDevOpsチームが管理する) データやアプリケーションから分離されます。モノリシックアプリケーションをマイクロサービスにリファクタリングするには、マイクロサービスを分離する必要があります。

# DevOps

コードの作成から本稼働までの迅速かつ協調的なワークフローを重視する、ソフトウェア開発と運用のアプローチ。DevOpsの目標は、機能を迅速に本番環境に展開するとともに、問題を検出してアプリケーションサービスを中断させることなく可能な限り短時間で修正することにあります。DevOpsでは、ほとんどの場合にアジャイルソフトウェア開発手法 (アジャイルソフトウェア開発の説明を参照) が用いられます。

# 分散型システム

サービスかノードのネットワークで動作するシステムまたはアプリケーション。分散型システムは、コンテナで実行されるマイクロサービスに適しています。

# Docker

アプリケーションとその依存関係を含む移植可能なコンテナの展開を自動化するためのオープンソースプロジェクト。2013年3月にリリースされたDockerは、それより前のLinuxコンテナ (LXC) のフォーマットをベースとしています。Dockerでは、アプリケーションはベアメタルからオンプレミスクラスター、パブリッククラウドインスタンスまでのさまざまなプラットフォーム上のコンテナ内で動作します。
Docker社は、以下のような複数の製品を生み出しています。

**Docker Compose**
マルチコンテナのDockerアプリケーションを定義して実行するためのツール。

**Docker Hub**
コンテナイメージを検索して共有するためのサービス。

**Docker Engine**
アプリケーションを構築してコンテナ化するためのオープンソースのコンテナ化テクノロジー。Docker Engineは、AIPとコマンドラインインターフェイスを使用してデーモンプロセスとして実行されるクライアントサーバーアプリケーションです。

**Docker Desktop**
開発者がコンテナ化されたアプリケーションとマイクロサービスを構築して共有できるMacまたはWindowsアプリケーション。Docker Desktopは、Docker Hubの認定されたコンテナイメージおよびテンプレートのライブラリへのアクセスを提供します。

# Git

ソースコードの変更を追跡するための分散型のバージョン管理システム。Linuxカーネルの開発用としてLinus Torvalds氏が2005年に作成したGitは、無料のオープンソースです。すべてのコンピューターのGitディレクトリはいずれも、包括的な履歴機能と完全なバージョン追跡機能を備える成熟したレポジトリです。

# JSON

人間が判読できるテキストを使用して属性値ペアとアレイデータタイプで構成されるデータオブジェクトを保存および転送する、オープン標準のファイルフォーマットとデータインタチェンジフォーマット。JSONは、当初2000年代の初期に規定され、2003年に標準化

されたJavaScriptから生まれた、非常に一般的な言語に依存しないデータフォーマットです。

# Kubernetes

当初Google社によって開発されたオープンソースのコンテナ管理/オーケストレーションソフトウェアシステム。Kubernetesは、コンテナのローンチとロードバランシングのために頻繁に使用されています。K8Sと略されることもあるKubernetesは、サービス検出、ロードバランシング、ヘルスチェック、コンテナレプリケーションといった、コンテナ化されたアプリケーションの展開、拡張、および管理を自動化します。DevOpsチームは、HTTP APIを使用してKubernetesにアクセスできます。

以下の8個の用語は、Kubernetesに関連するものです。

**Minikube**
VM内の単一のノードクラスターとしてKubernetesをローカルで実行する、広く普及している開発者ツール。

**kubectl**
Kubernetes APIサーバーと通信してKubernetesオブジェクトを作成、調査、更新、削除するためのコマンドラインツール。

**kubelet**
クラスター内の各ノード上のPodで動作するすべてのコンテナのヘルスチェックを行うためにそれらのノードで実行されるエージェント。kubeletはpodspecを読み取り、podspecに記載されているコンテナが動作していることと正常であることを確認します。

**Kubernetes APIサーバー**
RESTfulインターフェイスでKubernetesの機能をサポートし、クラスターの状態を保存するアプリケーション。Kubernetesのリソースはすべて APIオブジェクトとして保存され、APIサーバーに対するRESTfulコールで変更できるため、構成を宣言的に管理することが可能です。中核となるKubernetes APIは柔軟性が高く、拡張することでカスタムリソースをサポートできます。kube-apiserverによってサポートされるこのAPIは、実質的にKubernetesコントロールプレーンのフロントエンドの役割を果たします。

**kube-controller-manager**
コントローラーのプロセスを実行してクラスターの状態を監視し、望ましいクラスターの状態を維持するために必要に応じて変更を加えるか要求するコントロールプレーンコンポーネント。

**ノード**
仮想マシンの場合もあれば物理マシンの場合もあるKubernetesのワーカーマシン。ノードは、Podを実行するのに必要なローカルのデーモンとサービスを実行し、コントロールプレーンで管理されます。Kubernetesの初期バージョンでは、ノードはminionと呼ばれていました。

**Pod**
Kubernetesでオーケストレーションされる同じノード上に展開された1つ以上のコンテナのグループである、Kubernetes内の主要なソフトウェアの管理単位。

**UID**
オブジェクトを識別するKubernetesシステムで生成される文字列。Kubernetesクラスターのライフタイム全体で作成されるオブジェクトにはすべて、個別のUIDが割り当てられます。

## メッシュ (サービスメッシュ)

APIを使用してアプリケーションインフラストラクチャサービス間の膨大なネットワークベースの通信を管理する、構成可能な低レイテンシのインフラストラクチャレイヤー。サービスメッシュは、コンテナ化されたエフェメラルアプリケーションインフラストラクチャ間の通信が迅速、確実、かつ安全に行われるようにします。サービスメッシュには、一般的にサービス検出、ロードバランシング、オブザーバビリティ、トレーサビリティ、認証、承認が含まれます。

## メッセージング

アプリケーション間のメッセージ (イベント、要求、および返信について説明する特別にフォーマットされたデータ) の交換。各環境で理解する必要があるのは共通のメッセージングフォーマットとメッセージングプロトコルだけのため、メッセージングによってプログラムによるさまざまなプログラミング環境 (言語、コンパイラ、オペレーティングシステム) での通信が容易になります。

## メッセージングサーバー

他のアプリケーションとの間で送信されるメッセージを処理するミドルウェアアプリケーション。メッセージングサーバーは通常、メッセージを待ち行列に入れて優先順位を決定し、クライアントアプリケーションがこうした処理を行わなくて済むようにします。Apache Kafkaは、Publish/Subscribeモデルを使用する、一般的に用いられているオープンソースの耐久性に優れたメッセージングサーバーです。

## マイクロサービス

単一目的の自己完結型の実行プログラムであり、他のインスタンスやサービスとは別に実行されます。マイクロサービスは、他のサービスとは独立して要求を受け取って処理し、それらに対応する設計となっています。マイクロサービスは、他のサービスに影響を与えることなく更新できる疎結合の要素で構成されており、一般的にはマイクロサービスアーキテクチャーに組み込まれてソフトウェアアプリケーションを形成します。

## モノリシックアプリケーション

ユーザーインターフェイスとデータアクセスコードを単一のプラットフォーム上の単一のプログラムに組み込んだ、単一階層のソフトウェアアプリケーション。モノリシックアプリケーションは、自己完結型で他のソフトウェアから独立しています。このような従来のソフトウェアアプリケーションの作成手法は、急速にマイクロサービスアーキテクチャーに取って代わられつつあります。

## OAuth 2.0

インターネットユーザーが他のWebサイトに秘密の認証情報を提供することなく、Webサイトやアプリケーションからそれらのサイト上にある情報にアクセスできるようにする手法として広く使用されている、アクセス委任のオープン標準。OAuthは、リソースオーナーに代わってサーバーリソースへの委任された安全なアクセスをクライアントに提供し、クライアントが認証情報を共有することなくサードパーティにアクセスを許可できるようにします。OAuthは、HTTPと連携する設計となっており、パスワードではなくアクセストークンを使用します。2012年にOAuth 1.0に取って代わったOAuth 2.0は、Webアプリケーション、デスクトップアプリケーション、モバイルフォン、およびスマートデバイスに固有の承認フローを提供します。

## 永続コンテナストレージ

電源サイクル後もデータを保持するデータストレージデバイス。エフェメラルコンテナには永続ストレージがなく、プログラムの終了後も使用する必要のあるデータベースなどのステートフルアプリケーションとの連携の面で課題があるため、これはコンテナ化の重要な概念です (「それぞれに合わせたストレージとコンテナの最適化」を参照)。

## ローリングデプロイ

古いアプリケーションを新しいバージョンのアプリケーションに少しずつ置き換える展開戦略。その間、新しいバージョンと古いバージョンが機能やユーザーエクスペリエンスに影響を与えることなく共存します。コンテナでマイクロサービスとして実行されるアプリケーションのローリングデプロイにより、ダウンタイムを伴うことなくアプリケーションを1つずつ更新できます。新しいバージョンのマイクロサービスで問題が発生した場合は、一時的に元のバージョンに戻すことが可能です。

## 自己修復

正しく動作していないことを認識し、人間が介入しなくても必要な調整を行って自動的に正常な動作に戻すことができるデバイス、ソフトウェア、またはシステム。たとえば、Kubernetesコンテナオーケストレーターは、障害が発生したコンテナを交換するとともにkubeletのヘルスチェックに応答しないコンテナを停止し、それらが再び機能する状態になるまでクライアントにアドバタイズしません。

## サービス検出

サービスが必要とする機能を提供できる別のサービスのインスタンスを見つけ出す手法。一般的には、サービスがDNSルックアップを実行して別のサービスを見つけ出してから、コンテナオーケストレーションフレームワークが要求を受けることのできるサービスのインスタンスのリストを示します。

## 12ファクター手法

Heroku社の開発者が起草し、2011年に公開された、Software as a Serviceアプリケーションを構築するための手法。12ファクターは、Martin Fowler氏の著書である『Patterns of Enterprise Application Architecture』と『Refactoring』の影響を大きく受けたものであり、12ファクターアプリケーションの開発は、実行環境間のポータビリティの最大化、高度な自動化、および開発と本稼働間の相違の最小化につながっています。このアプリケーションは、ツール、アーキテクチャー、または開発手法を大幅に変更することなくプログラムで拡張できます。12ファクター手法は、あらゆるアプリケーションスタックの任意のプログラミング言語に適用することが可能です。

# ワークロード

アプリケーションを構成するすべてのプロセスとマイクロサービス。

# XML (Extensible Markup Language)

機械と人間の両方が判読できるドキュメントのコード化に関する一連のルールを定めるマークアップ言語。1998年にWorld Wide Web Consortiumが発表したXML 1.0は、シンプルさ、普遍性、およびさまざまなインターネットプラットフォームでのユーザビリティに重点が置かれています。XMLは、ユニコードを強く支持するテキストデータフォーマットです。XMLベースの言語は、スキーマで記述されます。XMLとJSONは、個別のアプリケーションプロセスを連携させるために頻繁にAPIで処理されます。

# YAML (YAML Ain't Markup Language)

さまざまなソフトウェアアプリケーションの入力形式として使用されるデータ指向の言語構造。YAMLは、すべてのプログラミング言語における人間が判読できるデータのシリアル化の標準です。