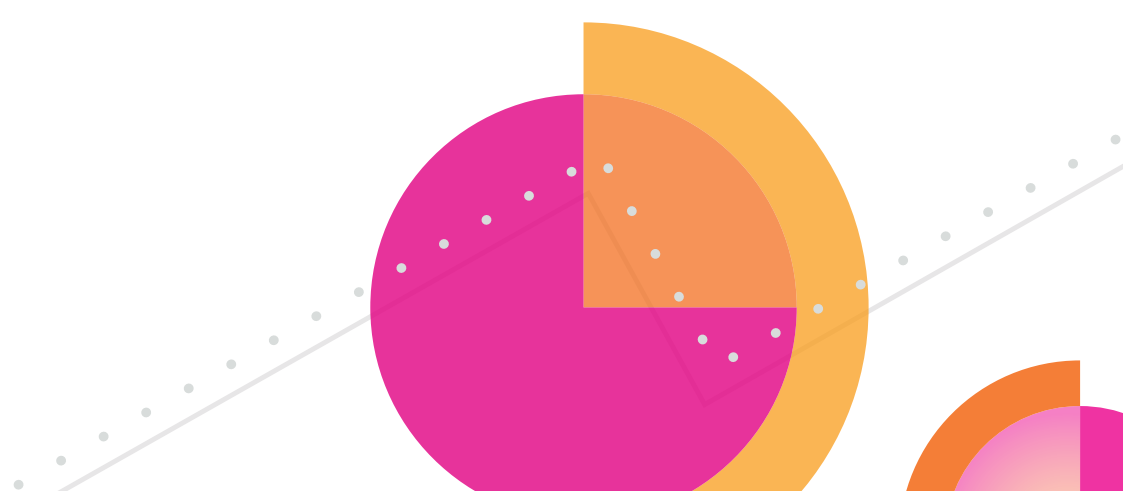
コンテナ監視とオーケストレーション

最新ガイド

splunk>
turn data into doing



コンテナは、2013年にその概念が紹介されて以来、IT業界で常に話題を呼んでいます。その理由は、アプリケーションコンテナテクノロジーが、アプリケーション開発プロセスの柔軟性と効率を飛躍的に高め、革命をもたらしているからです。

企業もこぞってコンテナを採用し始めています。ガートナー社の予測によると、コンテナ化されたアプリケーションを使用する企業は世界全体で、2019年の35%以下から2025年までには85%以上に増加する見通しです。この普及状況を考えると、今後企業が競争力を維持するには、コンテナベースの開発アプローチを採用することが欠かせないのは明白です。

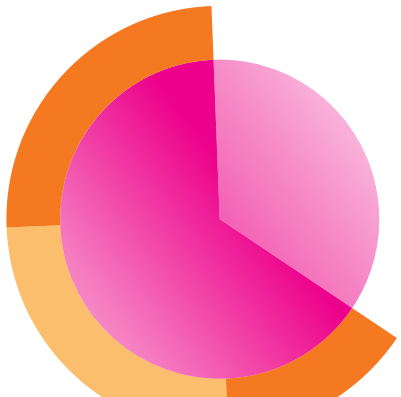
本書では、コンテナの基礎知識と競争優位を得るためのコンテナの活用方法について説明します。

コンテナの概要

コンテナの概念を理解する最も簡単な方法は、名前の由来を考えることです。物理的なコンテナは、中に物を入れて別の場所に運ぶための容器です。

ソフトウェアコンテナも機能はほぼ同じです。アプリケーションコード、設定ファイル、ライブラリ、システムツールなど、アプリケーションの実行に必要なすべてのものを1つの自己完結型のパッケージにまとめることで、どこにでも移動して実行できるようにします。

コンテナは「マイクロサービス」アプローチの重要コンポーネントです。このアプローチでは、アプリケーションを単一機能のモジュール単位に分割して、必要なときだけアクセスします。これにより開発者は、機能変更が必要になったときに、アプリケーション全体ではなく特定のサービスだけを修正して再デプロイできます。



コンテナが注目される理由

コンテナは、デプロイ環境に関係なくソフトウェアを常に一貫した動作で安定稼働させるという、運用において一般的な課題を解消します。アプリケーションをオペレーティングシステム、ネットワークトポロジー、セキュリティポリシーなどが異なる環境に移行すると(ステージング環境から本番環境に移行するなど)問題が起きることがありますが、コンテナを使用すれば、環境の違いを抽象化して、アプリケーションをその実行環境から切り離すことができます。複数のコンポーネントがバージョンの異なる共有ライブラリや依存コンポーネントを必要とする場合にも、コンテナならすべての依存コンポーネントを格納できるため、問題を回避できます。

コンテナが登場する前は、単一サーバー上で複数のアプリケーションを切り離して実行する方法として、仮想マシン(VM)が主に使われてきました。コンテナと同様にVMでも、コンピューターの基本インフラを抽象化することで、ハードウェアやソフトウェアの変更がアプリケーションの実行に影響を及ぼさないようにすることができます。ただし、それぞれの抽象化の方法には大きな違いがあります。

VMでは、物理サーバーを複数の仮想サーバーに分割することによってハードウェアを抽象化します。そのために、「ホストマシン」と呼ばれる物理コンピューター上でハイパーバイザーを実行し、そのハイパーバイザー上でVMを実行します。ハイパーバイザーは、基本的に、ホストマシンのリソース(CPU、RAMなど)へのアクセスを代行して「ゲストマシン」であるVMにリソースを提供する、仲介的な役割を担います。アプリケーションとその実行に必要なもの(ライブラリやシステムバイナリなど)はすべて、ゲストマシンにインストールします。また、オペレーティングシステムもゲストマシンごとにインストールします。そのため、たとえば4つのVMを実行するサーバーでは、4つのオペレーティングシステムと、それらの仲介役となるハイパーバイザーが動作することになります。1台のコンピューターのリソースを共有するため、処理量が増えるとパフォーマンスが急速に悪化します。その結果、1台のサーバーで実行できるVMの数が制限されます。

一方、コンテナでは、オペレーティングシステムレベルで抽象化が行われます。ホスト(物理サーバー、VM、またはクラウドホスト)上で1つのホストオペレーティングシステムを実行し、Docker Engineなどのコンテナエンジンを使用する複数のコンテナが、独立したユーザースペースを維持しながら、OSのカーネルを共有します。VMよりもオーバーヘッドがずっと少なく、軽量でリソースの使用効率が良いため、サーバーリソースをはるかに有効活用できます。

コンテナを使用する5つのメリット

コンテナベースのインフラには多くのメリットがあります。
その代表的なものを5つご紹介します。

1. **提供の速度**：仮想マシンにインストールしたアプリケーションは、通常、起動に数分かかります。コンテナではオペレーティングシステムの起動を待つ必要がないため、アプリケーションは瞬時に起動します。また、ホストOSのリソース消費量が少ないため動作が速く、作成、クローン作成、破棄は数秒で完了します。これらの高速性は特に、ソフトウェアのリリース、バグ修正、新機能追加の期間短縮に役立ち、開発プロセスに大きなメリットをもたらします。

2. **DevOpsファーストのアプローチ**：マイクロサービスベースのコンテナの速さ、フットプリントの小ささ、リソース効率の高さは、DevOps環境に理想的です。マイクロサービスベースのインフラでは、開発者がアプリケーションの特定の機能を端から端まで担当するため、その動作を完全に把握し、モノリシックなアプリケーションの場合よりも効率的にパフォーマンスの最適化とトラブルシューティングが行えます。

3. **移行性**：コンテナにはアプリケーションとそのすべての依存コンポーネントが含まれます。そのため、Windows、Linux、Macのいずれのハードウェア間でも簡単に移行でき、安定稼働します。また、ベアメタル、仮想サーバー、パブリッククラウド、プライベートクラウドのいずれの基盤でも動作します。特定のパブリッククラウドから別の環境にアプリケーションを簡単に移行できるため、ベンダーロックインの回避にも役立ちます。

4. **拡張性**：コンテナは、VMとは異なり独自のOSが不要であるため、サイズが小さめです。一般的なコンテナのサイズは1つあたり数十メガバイトで、数十ギガバイトにも及ぶVMの約1,000分の1です。そのため、1つのホストオペレーティングシステムに多数のコンテナを格納でき、拡張性が向上します。

5. **一貫性**：コンテナではすべての依存コンポーネントと設定が1カ所にまとめられるため、開発者は、コンテナのデプロイ先に関係なく一貫した環境で作業できます。これにより、環境の違いによって生じる問題の解決に時間をとられることなく、アプリケーションの新機能の開発に集中できます。また、本稼働が決まったときは、開発環境から本番環境にコンテナをそのまま移行できます。さらに、コンテナは一度作成すると変更されないため、開発者は、デプロイ中に設定が変わるなど、トラブルシューティングを難しくする問題から解放されます。

オーケストレーションの基礎：Kubernetesの使用

コンテナのオーケストレーションを開始するには、コンテナ化したアプリケーションをデプロイ、管理、スケーリングするための専用のソフトウェアが必要です。その中で、定評があり今日でも人気が高いのがKubernetesです。Kubernetesは、Google社が開発し、現在はCloud Native Computing Foundationが管理している、オープンソースの自動化プラットフォームです。

Kubernetesを使えば、開発プロセスを劇的に向上させることができます。コンテナ管理を効率化し、更新とスケーリングを自動化して、ダウンタイムを最小限に抑えることで、開発者はアプリケーションの機能改善と新機能の開発に集中できます。その仕組みを理解するために、Kubernetesの基本コンポーネントとそれらの相互関係について説明します。

Kubernetesでは、独自の用語で定義される複数の抽象化レイヤーが使用されます。Kubernetesは多くのコンポーネントで構成されます。ここではすべてを取り上げませんが、システム内でハードウェアとソフトウェアがどのように表現されるかについて簡単に説明します。

ノード：Kubernetes用語では、単一のすべての「ワーカーマシン」をノードといいます。ノードには、物理サーバーだけでなく、アマゾン ウェブ サービス(AWS)やMicrosoft Azureなどのクラウドプロバイダーが提供する仮想マシンも含まれます。当初はノードを「ミニオン」(子分)と呼んでいました。その名のとおり、ノードはマスターノードから割り当てられたタスクを受信して実行します。ノードには、リソースを管理してコンテナに割り当てるために必要なすべてのサービスが含まれます。

マスターノード：すべてのワーカーノードを管理し、ユーザーがKubernetesを操作するために使用するマシンです。タスクはすべてここから割り当てられます。

クラスター：マスターノードとその配下のワーカーノードが1つのクラスターを構成します。クラスターは、これらすべてのマシンを1つの強力なユニットとして統合します。コンテナ化されたアプリケーションはクラスターにデプロイされ、クラスターが各ノードにワークロードを分散させます。また、ノードが追加または削除されたときは、クラスターがワークロードを移動します。

Pod：グループ化されてノードにデプロイされたコンテナをまとめてPodといいます。同じPod内のコンテナは、ローカルのネットワークやその他のリソースを共有します。また、同じマシン上にある場合と同様に相互に通信できますが、互いに切り離された状態は維持されます。同時に、Podによって、ネットワークとストレージも基盤となるコンテナから切り離されます。

1台のワーカーノードに複数のPodをデプロイできます。いずれかのノードで障害が発生したときは、正常なノードに代替のPodがデプロイされます。

Podには多数のコンテナをデプロイできますが、必要なものだけ、つまりメインプロセスを実行するコンテナとそれを補佐するコンテナ、まとめて「サイドカー」と呼ばれる単位でデプロイすることをお勧めします。Podは、コンテナの個々のニーズに関係なく、1つの単位としてスケーリングされるため、コンテナが過剰にあるPodは、リソースを大量に消費することがあります。

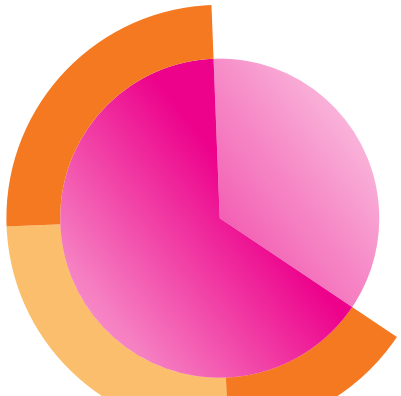
Deployment：Kubernetesでは、Podをクラスターに直接デプロイするのではなく、「Deployment」と呼ばれる追加の抽象化レイヤーを使用します。Deploymentでは、Podごとに、同時実行するレプリカ数を指定できます。Deploymentは、指定された数のPodをクラスターにデプロイした後、それらを監視して、いずれかに障害が発生したときには自動的にPodを再作成して再デプロイします。

Ingress：KubernetesではPodが外部から切り離されるため、外部のサービスからPodにアクセスするには、通信チャネルを確立する必要があります。そのために使用するのが、もう1つの抽象化レイヤーである「Ingress」です。クラスターにIngressを追加するには、LoadBalancer、NodePort、またはIngressコントローラーを追加するなど、いくつかの方法があります。Ingressは、従来のアーキテクチャでインターネット側の窓口としてよく使われるWebサーバーと同じような役割を果たします。

Kubernetesとコンテナが監視にもたらす課題

コンテナとオーケストレーションフレームワークは多くのメリットをもたらしますが、同時に、クラウドベースアプリケーションの管理が複雑になるというデメリットももたらします。コンテナには以下のような課題があります。

- **大きな盲点が生まれる：**コンテナは使い捨てできるように設計されています。そのために、アプリケーションとその基盤となるハードウェアの間にいくつかの抽象化レイヤーを挟むことによって、可搬性と拡張性を確保しています。こうした設計が、従来の監視方法では大きな盲点を生みます。従来の監視ツールでは、抽象化レイヤーがあると環境全体を把握できなくなります。



- **管理するデータが増える：**多くの独立したコンポーネントを簡単に移動できるため、アプリケーション、コンテナ、オーケストレーションプラットフォームのパフォーマンスや信頼性を常時監視するには、テレメトリデータを継続的に管理する必要があります。また、マイクロサービスアーキテクチャでは、通常、必要に応じてマイクロサービスを拡張し、不要になったら破棄します。このエフェメラル(揮発的)な性質も、オブザーバビリティシステムに取り込むデータを増やす原因になります。さらに、システムにコンポーネントを追加することで、監視対象が増え、問題が発生したときの確認範囲が拡大します。

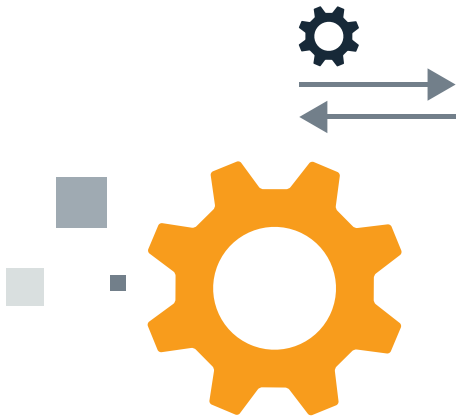
- **可視化の重要性が高まる：**マイクロサービス、コンテナ、コンテナオーケストレーションを導入すると、規模が大きくなり複雑さが増します。そのため、環境全体を可視化して、インフラの健全性に関するインサイトをすばやく獲得するとともに、環境内のトラフィックの流れを把握できるようにする必要があります。また、コンテナ、ノード、Podの健全性とパフォーマンスを詳細に確認できるようにすることも大切です。このワークフローを実現するには、それに対応する監視ソリューションが必要です。

- **DevOpsのペースと合わない：**コンテナは瞬時にスケーリングや変更が可能です。デプロイのペースが上がると、DevOpsチームにとって、全体としてアプリケーションのパフォーマンスにどのような影響があるかを追跡するのが難しくなり、さらにはサービスの依存関係が新たに追加されても把握できない可能性があります。

コンテナの導入

優れたコンテナ監視ソリューションを導入すれば、コンテナデータを他のインフラデータと統合して、コンテキストを明確にし、根本原因分析の精度を上げることによって、コンテナベースの動的な環境を適切に管理できます。一般的な実装であるDockerを例に、優れたソリューションがどのように複数の監視レイヤーを実現するかをご紹介します。

ホスト：クラスター内の物理マシンと仮想マシンの可用性およびパフォーマンスを監視できます。追跡対象となる主なメトリクスには、メモリー、CPU使用率、スワップ使用領域、ストレージ使用率などが含まれます。これは、どのコンテナ監視ツールでも中核となる機能です。



コンテナ：コンテナを全体および個別に可視化することは重要です。監視ツールでは、現在実行中のコンテナの数、メモリーを最も多く消費しているコンテナ、最近起動したコンテナに関する情報を確認できます。また、各コンテナのCPUやメモリーの使用率、ネットワークI/Oの健全性に関するインサイトも提供されます。

オーケストレーションフレームワーク：Kubernetes自体も監視する必要があります。利用可能なノードはいくつあるか、マスターノードは正常に稼働しているか、再割り当てが長期間保留になっているPodはないか、Ingress経由のトラフィックはどのくらいあるか、などを把握します。高い信頼性でサービスを運用し続けるには、これらの状況をすばやく確認できなければなりません。また、Kubernetesでは基本的に、リソース使用率が最適になるようにPodがスケジュールされます。これにより効率が向上しますが、Podのデプロイと実行状況を予測できないという問題もあります。

アプリケーションエンドポイント：サービスがユーザーリクエストを処理できる状態にあるかを把握し、リクエストのパフォーマンスと遅延を測定することも非常に重要です。監視ソリューションでアプリケーション自体の健全性チェックを実行して、遅延やその他のパフォーマンスメトリクスを測定する必要もあります。

コンテナ化したアプリケーションの監視に必要な機能

前述のとおり、アプリケーションのコンテナ化と、Kubernetesのようなオーケストレーションフレームワークの導入は、最新の開発/デプロイワークフローに大きなメリットをもたらします。一方で、このような環境とアプリケーションを監視することは、従来のツールでの監視よりもはるかに複雑になります。コンテナとオーケストレーションワークフローを適切に監視するには、監視ソリューションに以下の機能が必要です。

重要なメトリクスの収集

Podのメトリクス：必要なPod数、利用可能なPod数、フェーズごとのPod数(障害発生、保留、実行中)、デプロイごとの必要なPod数

リソース使用状況のメトリクス：DockerのSocketで収集されるメトリクス(CPUやメモリーの使用率など、コンテナレベルとノードレベルのリソースメトリクス)

アプリケーションのメトリクス：REDメトリクス(リクエスト数、エラー数、処理時間)、アプリケーションの健全性、データベースの可用性とパフォーマンス

統合、相関付け、分析

簡単導入：コレクターの導入はクラウドネイティブで簡単に実行できることが大切です。Helmチャートを利用できるのが理想的です。Splunkなら以下のように簡単に導入できます。

```
helm repo add splunk-otel-collector-chart https://
signalfx.github.io/splunk-otel-collector-chart
```

```
helm repo update
```

```
helm install --set splunkAccessToken='xxx' --set
clusterName='sample' --set splunkRealm='us0' --set
otelCollector.enabled='true' --generate-name splunk-
otel-collector-chart/splunk-otel-collector
```

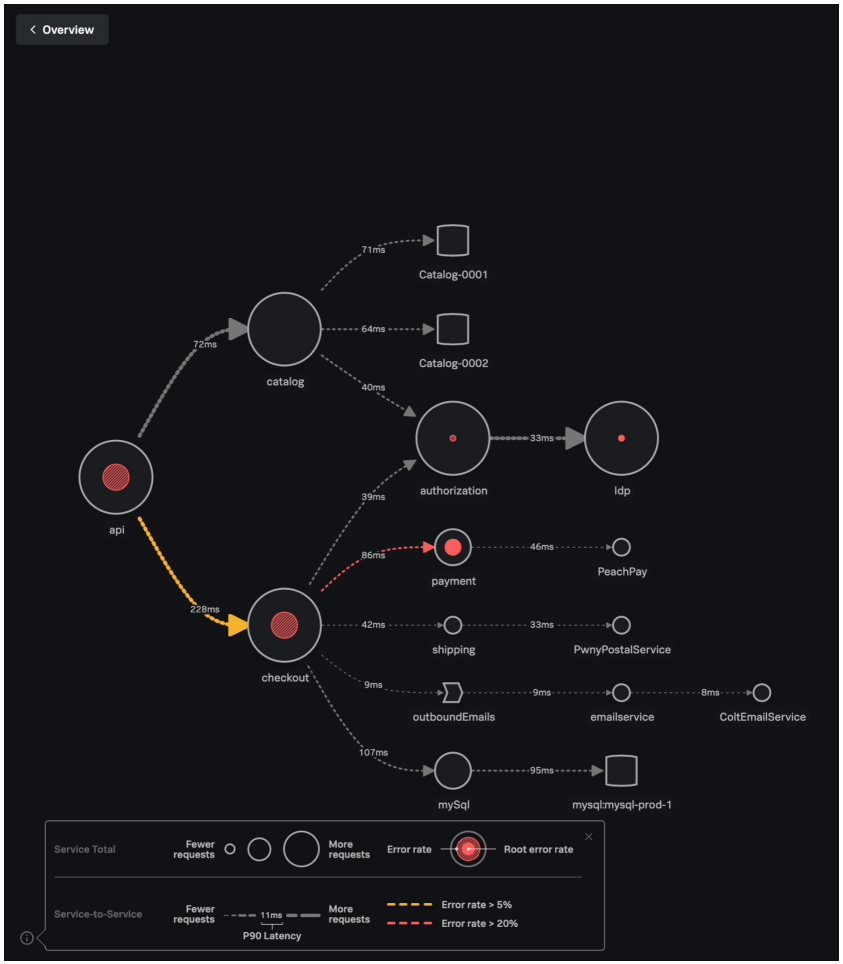
ロックインの回避：OpenTelemetryは注目される監視テクノロジーです。環境のインストルメンテーション(計装)では、監視ツールやオブザーバビリティ(可観測性)ツールのプロバイダーを変更する可能性を考慮することが重要です。ソリューションを選ぶときは、ツールを変更することになってもインストルメンテーションをやり直す必要がないよう、OpenTelemetryをサポートしているかどうかを確認しましょう。

リアルタイムのストリーミングプラットフォーム：1時間のダウンタイムが数十万ドルの損失につながる今日では、数秒でもサービスを停止したくありません。最新の監視/オブザーバビリティプラットフォームでは、アラート生成とデータ分析がリアルタイムで実行されるかどうかが重要なポイントです。リアルタイムで処理できれば、MTTD (平均検出時間)を最小限に抑え、システムの最新の状態を示すデータに基づいて状況を判断できます。

予測分析：AIと機械学習を活用した予測分析ツールがあれば、障害を未然に察知できます。最新の複雑な環境でも、問題を発生前に解消し、顧客に影響が及ぶ前にパフォーマンスや可用性の低下を検出できます。

事前構築済みのダッシュボード：インフラの構成を完全に把握していなくても監視できることも重要です。自動化された内蔵のダッシュボードがあれば、ノード、Pod、コンテナ、アプリケーション間の複雑な相互関係を簡単に理解し、環境全体の状況を一目で把握できます。

サービスマップの自動作成：最新の環境では、Ingress、コンテナ、アプリケーションでのトラフィックの流れを理解するのが非常に困難です。リクエストのパスを追跡して表示し、環境内で発生したエラーやその他の問題を検出して警告する機能があれば、監視が容易になります。



次のステップ

コンテナは開発プロセスを向上させる強力なツールですが、コンテナ環境の状態を常に把握し、改善することが欠かせません。コンテナの導入後は、インフラ監視とアプリケーションパフォーマンス監視の重要性が増します。コンテナに対応した監視ソリューションに求められる機能には、内蔵のコンテナ監視、簡単導入、リアルタイムのストリーミングプラットフォーム、予測分析、有意義なデータをすぐに活用できるエクスペリエンスなどがあります。これらすべての機能を備え、コンテナ、パブリッククラウド、プライベートクラウド、自社ホスト環境でネイティブに動作するオブザーバビリティシステムの導入をご検討であれば、Splunk Observability Cloudのデモまたは無料トライアル版をお試しください。オブザーバビリティについて詳しくは、SplunkのWebサイトまたは『オブザーバビリティ (可観測性)ビギナーガイド』をご覧ください。

splunk>

21-14769-Splunk-ModernGuidetoContainerMonitoringandOrchestration-EB-JA-202203