RED HAT®
CONSULTING

redhat.

ホワイトペーパー

コンテナベース・アプリケーションの設計原則

Bilgin Ibryam 著

エグゼクティブサマリー

「クラウドネイティブ」とは、クラウドベースのインフラストラクチャでの実行を想定して設計されたアプリケーションを示す用語です。一般には、クラウドネイティブ・アプリケーションは、プラットフォームで管理されるコンテナ内で実行される、疎結合されたマイクロサービスとして開発されます。これらのアプリケーションは障害の発生をあらかじめ想定しており、基盤のインフラストラクチャに障害が発生しても、安定して実行およびスケーリングできます。このような機能を実現するため、クラウドネイティブ・プラットフォームは、その上で実行されるアプリケーションに一連のコントラクトと制約を課します。このコントラクトによって、アプリケーションを特定の制約に準拠させ、プラットフォームはコンテナ化アプリケーションの管理を自動化できます。

多くの組織が、クラウドネイティブ化の必要性と重要性を理解しつつも、どこから着手すべきかわからずにいます。クラウドネイティブ・プラットフォームとその上で実行されるコンテナ化アプリケーションをシームレスに連携させると、障害を予測する能力と、基盤のインフラストラクチャに障害が発生しても実行と拡張が可能な信頼性が得られます。このホワイトペーパーでは、コンテナ化アプリケーションを優れたクラウドネイティブ・アプリケーションとするために守るべき原則を説明します。これらの原則は、Kubernetes などのクラウドネイティブ・プラットフォームでの自動化に適したアプリケーションの設計を助けます。

ソフトウェア設計の原則：

- Keep it simple, stupid (KISS) (シンプルにしておく)
- Don't repeat yourself (DRY) (重複を防ぐ)
- You aren't gonna need it (YAGNI) (機能は必要になるまで追加しない)
- Separation of concerns (SoC) (問題の分離)

クラウドネイティブ・コンテナに対する RED HAT のアプローチ：

- Single concern principle (SCP) (単一関心の原則)
- High observability principle (HOP) (高観測可能性の原則)
- Life-cycle conformance principle (LCP) (ライフサイクル適合の原則)
- Image immutability principle (IIP) (イメージ不変性の原則)
- Process disposability principle (PDP) (プロセス廃棄性の原則)
- Self-containment principle (S-CP) (自己完結性の原則)
- Runtime confinement principle (RCP) (ランタイム制限の原則)

facebook.com/redhatjapan
@redhatjapan
linkedin.com/company/red-hat

jp.redhat.com

一般的なコンテナ関連のベストプラクティス：

- イメージは小規模になるよう目指す
- 任意のユーザー ID をサポートする
- 重要なポートをマークする
- 永続データにボリュームを使用する
- イメージにメタデータを設定する
- ホストとイメージを同期させる

ソフトウェア設計の原則

原則は生活の中のさまざまな場面に存在し、一般には基本的な真実や信念を表し、ここからまた別の原則が導き出されます。ソフトウェアの場合、原則はどちらかと言えば抽象的なガイドラインで、ソフトウェア設計時に従うべきものと考えられています。あらゆるプログラミング言語に適用され、異なるパターンを使用して実装し、さまざまな実践手法に従って実行できます。

通常は、パターンと実践手法は、原則から期待される成果を達成するために使用されるツールです。高品質のソフトウェアを作成する基本的な原則があり、これらからその他すべての原則が導かれています。このような原則の例を示します。

- **KISS** - Keep it simple, stupid (シンプルにしておく)
- **DRY** - Don't repeat yourself (重複を防ぐ)
- **YAGNI** - You aren't gonna need it (機能は必要になるまで追加しない)
- **SoC** - Separation of concerns (問題の分離)

これらの原則は具体的なルールを示すものではありませんが、多くの開発者が理解し、たびたび取り上げる表現や共通の知恵を表現しています。

この他にも、SOLID (**S**ingle responsibility (単一責任)、**O**pen/closed (オープン/クローズ)、**L**iskov substitution (Liskov 置換)、**I**nterface segregation (インタフェース分離)、**D**ependency inversion (依存性の逆転) の原則があります。これは Robert C. Martin 氏が提唱したもので、より良いオブジェクト指向ソフトウェアを作成するためのガイドラインを示すものです。汎用の補完的な原則からなるフレームワークで、さまざまな解釈の余地はありつつも、優れたオブジェクト指向設計を行うためには十分な指針となります。SOLID の原則を適用すると、品質が高くて長期的に保守が容易になるシステムを作成できる可能性が高まると予期されます。

SOLID の原則は、クラス、インタフェース、継承などのオブジェクト指向のプリミティブとコンセプトをオブジェクト指向設計の論拠に使用しています。同様に、クラウドネイティブ・アプリケーションの設計の原則もあり、主なプリミティブはクラスではなくコンテナイメージになっています。これらの原則に従うと、Kubernetes などのクラウドネイティブ・プラットフォームへの適合性が高いコンテナ化アプリケーションを作成できる可能性が高まります。

クラウドネイティブ・コンテナに対する RED HAT のアプローチ

現在、ほとんどのアプリケーションはコンテナに格納して実行することが可能です。しかし、Kubernetes などのクラウドネイティブ・プラットフォームで効率的に自動化およびオーケストレーションできるコンテナ化アプリケーションを作成するには、追加の労力が必要です。

以下の内容は、ソースコード管理からアプリケーションのスケーラビリティ・モデルまでを取り扱う「Twelve-Factor App」などの他の多くの業績にヒントを得たものです。ただし、以下の原則の対象範囲は、Kubernetes などのクラウドネイティブ・プラットフォーム向けのコンテナ化されたマイクロサービスベースのアプリケーションの設計に限定されています。

コンテナ化アプリケーションの作成に関わる以下の原則では、コンテナイメージを基本のプリミティブとして、コンテナ・オーケストレーション・プラットフォームをターゲットのコンテナ・ランタイム環境として使用しています。これらの原則に従うと、作成されたコンテナは、ほとんどのコンテナ・オーケストレーション・エンジンにおいてクラウドネイティブに動作し、自動化された方法でスケジュール、拡張、監視できるようになります。これらの原則の記載順序に特別な意味はありません。

SINGLE CONCERN PRINCIPLE (SCP) (単一関心の原則)

この原則は多くの意味で、クラスはただ 1 つの責任を持つべきだという、SOLID の単一責任の原則 (SRP) に類似しています。SRP の背景となる意図は、各責任は変化の軸であり、クラスに必要な変更理由はただ 1 つだけだということです。SCP 原則の「concern (関心)」は、関心を責任ではなく高レベル

の抽象化として捉え、スコープをクラスではなくコンテナとして記述しています。SRP の主な意図は変化の理由は 1 つだけに限定することですが、SCP の主な意図はコンテナイメージの再利用と置換可能性です。1 つの関心に対処するコンテナを作成し、実装完了の方法を採用した場合、コンテナイメージを別のアプリケーションコンテキストで再利用できる可能性が高まります。

このため、SCP 原則では、各コンテナが 1 つの関心に対処し、それもうまく対処すべきだと規定しています。この原則の達成は、オブジェクト指向の世界での SRP の達成よりも簡単です。コンテナは通常 1 つのプロセスを管理し、多くの場合、この 1 つのプロセスが 1 つの関心に対処するからです。

図 1. 1 つのデプロイユニット内にある複数の単一関心コンテナ

コンテナ化されたマイクロサービスで複数の関心に対処する必要がある場合、サイドカーなどのパターンや Init Container を使用して、複数のコンテナを 1 つのデプロイユニット (Pod) にまとめられます。ここで、各コンテナは 1 つの関心を処理できます。同様に、同じ関心に対処するコンテナを交換できます。たとえば、Web サーバーコンテナやキュー実装コンテナを、新しく拡張性の高いコンテナに置換できます。

HIGH OBSERVABILITY PRINCIPLE (HOP) (高観測可能性の原則)

コンテナは、アプリケーションをブラックボックスのように取り扱い、統一された方法でパッケージ化して実行します。しかし、コンテナをクラウドネイティブ化するためには、ランタイム環境にアプリケーション・プログラミング・インタフェース (API) を提供することでコンテナの正常性を監視し、状態に応じて動作させる必要があります。これはコンテナのアップデートとライフサイクルを統一された方法で自動化するための基本的な前提条件で、これによってシステムの回復力とユーザーエクスペリエンスが改善されます。

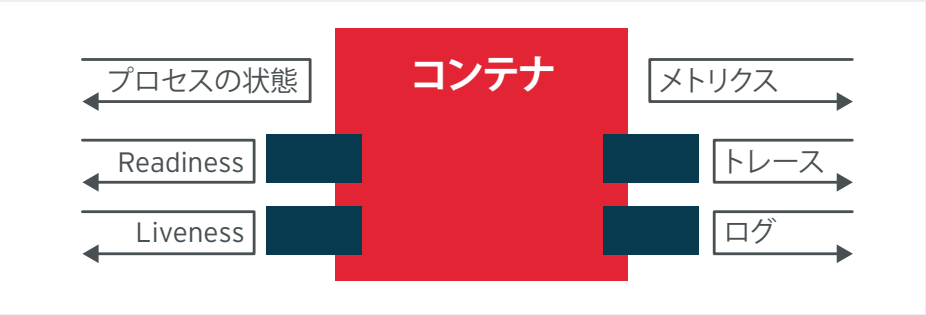

図 2. 複数の観測可能性 API を持つコンテナ (コンテナには観測可能性を有効にする複数の API がある)

実践面ではコンテナ化アプリケーションは、活動状況や準備状況など、さまざまな状態チェックに対して API を提供することが最低条件になります。適切に動作するアプリケーションでも、その他の手段を提供してコンテナ化アプリケーションを確認する必要があります。アプリケーションは重要なイベントを標準エラー (STDERR) と標準出力 (STDOUT) に記録して、Fluentd や Logstash などのツールでログを集約し、OpenTracing や Prometheus などのトレースおよびメトリクス収集ライブラリと統合する必要があります。

アプリケーションをブラックボックスとして扱いながら、必要なすべての API を実装してプラットフォームがアプリケーションを最適な方法で監視して管理できるようにします。

LIFE-CYCLE CONFORMANCE PRINCIPLE (LCP) (ライフサイクル適合の原則)

HOP では、プラットフォームが情報を読み取る API をコンテナで提供することが求められます。LCP の意義は、プラットフォームから送信されるイベントを読み取る方法をアプリケーションに持たせることです。さらに、イベントを取得する他に、コンテナはこれらのイベントに従って応答する必要があります。これが、この原則の名前の由来です。アプリケーションに「書き込み API」を用意してプラットフォームとやり取りするのと、ほぼ同じです。

図 3. API を提供し、プラットフォームイベントに従うコンテナ

管理するプラットフォームからはあらゆる種類のイベントが送信されます。これらのイベントはコンテナのライフサイクル管理を支援するためのものです。どのイベントを処理するかや、イベントに応答すべきかどうかは、アプリケーション側で決定します。

ただし、一部のイベントには他のイベントよりも高い重要性があります。たとえば、クリーン・シャットダウン・プロセスを必要とするアプリケーションは、終了 (SIGTERM) メッセージというシグナルをキャッチしたら、即座にシャットダウンする必要があります。これは、SIGTERM に続くキル (SIGKILL) シグナルによる強制シャットダウンを防ぐためです。

この他にも、PostStart や PreStop など、アプリケーションのライフサイクル管理にとって重要なイベントがあります。たとえば、一部のアプリケーションはサービス要求の前にウォームアップが必要で、別のアプリケーションはクリーンシャットダウンの前にリソースを解放する必要があります。

IMAGE IMMUTABILITY PRINCIPLE (IIP) (イメージ不変性の原則)

コンテナ化アプリケーションは不変であるため、ビルドされた後、異なる環境間で変化することは想定されていません。つまり、環境ごとにコンテナの作成や修正を行うのではなく、ランタイムデータの保存に外部手段を利用し、外部化した設定を環境によって使い分けます。コンテナ化アプリケーションを変更すると、新しいコンテナイメージが作成され、このイメージがすべての環境で再利用されることになります。同じ原則はイミュータブル・サーバー/インフラストラクチャとして知られており、サーバー/ホスト管理にも使用されています。

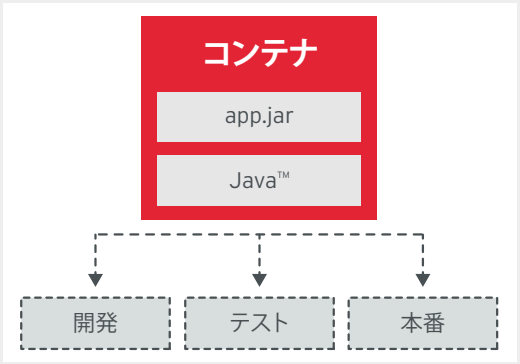

図 4. すべての環境に不変コンテナイメージが使用される

IIP の原則に従うことで、異なる環境向けの類似したコンテナイメージの作成を防ぎ、1 つのコンテナイメージを各環境向けに設定できるようになります。この原則によって、アプリケーションのアップデート中における自動ロールバックやロールフォワードなどの手法が可能になります。これは、クラウドネイティブ自動化にとって重要な観点と言えます。

PROCESS DISPOSABILITY PRINCIPLE (PDP) (プロセス廃棄性の原則)

コンテナ化アプリケーションに移行する主な動機の 1 つは、コンテナは可能な限り一時的なものとし、いつでも別のコンテナのインスタンスと交換できなければならないことです。コンテナを置き換える理由は、ヘルスチェックの不合格、アプリケーションのスケールダウン、別のホストへのコンテナの移行、プラットフォームリソースの枯渇やその他の問題など、多数あります。

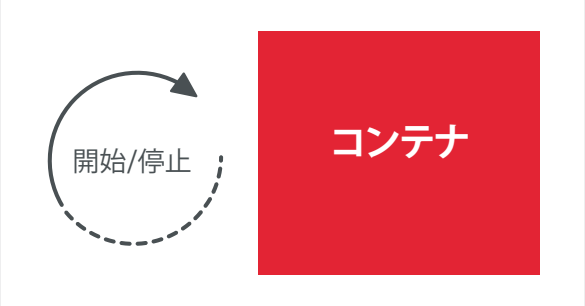

図 5. すばやい起動とシャットダウンで交換が簡単なコンテナ化アプリケーション

これは、コンテナ化アプリケーションは状態を外部化または分散させ、冗長性を維持しておく必要があることを意味します。また、アプリケーションは起動やシャットダウンを速やかに行い、急なハードウェア故障にも対処できなければならないということでもあります。

この原則の実装に役立つもう 1 つの手法は、小型のコンテナを作成することです。クラウドネイティブ環境にあるコンテナは、自動的なスケジューリングのもと、別のホストで起動される場合があります。コンテナを小型化すると、起動時間が短くなります。コンテナは、再起動する前にホストシステムに物理的にコピーする必要があるためです。

SELF-CONTAINMENT PRINCIPLE (S-CP) (自己完結性の原則)

この原則の意味は、ビルド時にコンテナは必要なすべてを含む必要があるということです。コンテナは Linux® カーネルのみを使い、追加のライブラリはすべてコンテナのビルド時に追加する必要があります。ライブラリの他にも、言語ランタイム、アプリケーション・プラットフォーム (必要な場合) など、コンテナ化アプリケーションの実行に必要なその他の依存関係をすべて含める必要があります。

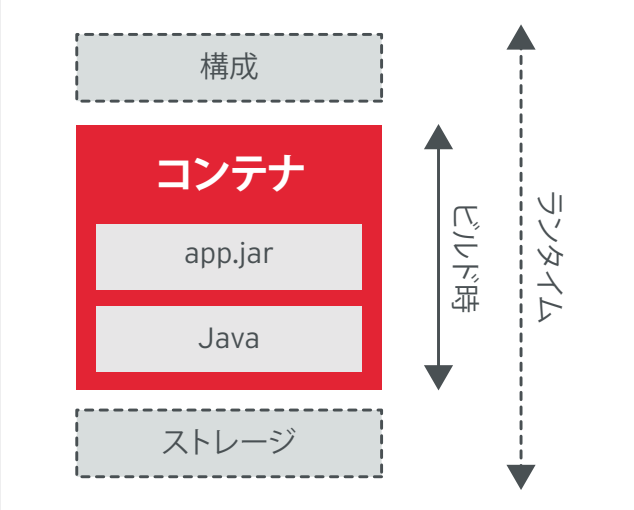

図 6. コンテナはビルド時に、環境固有の関係以外のすべての依存関係を組み込む

唯一の例外は、設定など環境に応じて異なるもので、これは Kubernetes ConfigMap などによってランタイム時に提供する必要があります。

一部のアプリケーションは、複数のコンテナ化されたコンポーネントで構成されます。たとえば、コンテナ化 Web アプリケーションにはデータベースコンテナも必要となることがあります。この原則では両方のコンテナの結合を推奨しません。結合するのではなく、データベースコンテナにはデータベースの実行に必要なすべてを格納し、Web アプリケーションコンテナには Web アプリケーションの実行に必要なすべてを格納します (Web サーバーなど)。ランタイム時に、Web アプリケーションコンテナは必要に応じてデータベースコンテナにアクセスして利用します。

RUNTIME CONFINEMENT PRINCIPLE (RCP) (ランタイム制限の原則)

S-CP はコンテナを、ビルド時と、ビルドによって作成されるバイナリーとそのコンテンツの観点から捉えています。しかしコンテナは、ディスク上の同一サイズの 1 次元のブラックボックスではありません。コンテナはランタイム時に、メモリー使用率の次元、CPU 使用率の次元、その他のリソース消費の次元など、複数の次元を持ちます。

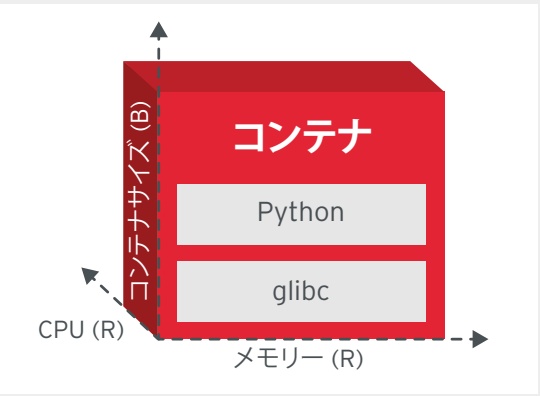

図 7. ランタイムリソース要件を宣言し、ランタイム時に要件に従うコンテナ

RCP 原則に従うと、各コンテナはリソース要件を宣言し、この情報をプラットフォームに渡すことになります。CPU、メモリー、ネットワーク、プラットフォームのスケジュール実行によるディスクへの影響、自動スケーリング、キャパシティ管理、コンテナの一般的なサービスレベル・アグリーメント (SLA) の観点で、コンテナのリソースプロファイルを共有します。

コンテナのリソース要件を渡す他にも、指定されたリソース要件内にアプリケーションが維持されることも重要です。アプリケーションが制限内に留まっていると、リソースが枯渇したときにプラットフォームによって終了されたり移行されたりする可能性が低くなります。

まとめ

クラウドネイティブとは、到達すべき状態ではなく、作業方法です。このホワイトペーパーでは、コンテナ化アプリケーションを優れたクラウドネイティブ・アプリケーションとするための方法を示す原則を紹介しました。

これらの原則に加えて、優れたコンテナ化アプリケーションを作成するには、他のコンテナ関連のベストプラクティスやテクニックに習熟する必要があります。上述した原則は基本的なもので、ほとんどのユースケースに適用されるものですが、以下のベストプラクティスにはてきようすべきかどうか判断が必要です。以下は、比較的一般的なコンテナ関連のベストプラクティスの例です。

- **イメージは小規模になるよう目指す：**一時ファイルをクリーンアップして小型のイメージを作成し、不要なパッケージのインストールを防止します。これにより、コンテナサイズ、ビルド時間、コンテナイメージのコピーにかかるネットワーク時間が短縮されます。
- **任意のユーザー ID をサポートする：**sudo コマンドの使用や、コンテナの実行に特定の userid を要求することは避けます。
- **重要なポートをマークする：**ランタイム時にポート番号を指定することはできますが、EXPOSE コマンドを使用して指定するほうが、ユーザーとソフトウェアの双方がイメージを簡単に使用できるようになります。
- **永続データにボリュームを使用する：**コンテナが破壊された後も保持する必要があるデータは、ボリュームに書き込む必要があります。
- **イメージにメタデータを設定する：**イメージにタグ、ラベル、注釈などのメタデータが設定されていると、コンテナイメージを利用しやすくなり、イメージを使用する開発者から見たエクスペリエンスも向上します。
- **ホストとイメージを同期させる：**一部のコンテナ化アプリケーションは、時間やマシン ID などの特定の属性について、コンテナとホストの同期を要求します。

以下はパターンとベストプラクティスに関する資料の一覧です。上記の原則を効果的に実装するためにお役立てください。

- https://www.slideshare.net/luebken/container-patterns
- https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices
- http://docs.projectatomic.io/container-best-practices
- https://docs.openshift.com/enterprise/3.0/creating_images/guidelines.html
- https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_burns.pdf
- https://leanpub.com/k8spatterns/
- https://12factor.net/

アジア太平洋 +65 6490 4200
オーストラリア 1 800 733 428
ブルネイ / カンボジア 800 862 6691
インド +91 22 3987 8888
インドネシア 001 803 440224
日本 03 5798 8510
韓国 080 708 0880
マレーシア 1 800 812 678
ニュージーランド 0800 450 503
フィリピン 800 1441 0229
シンガポール 800 448 1430
タイ 001 800 441 6039
ベトナム 800 862 6691
中国 800 810 2100
香港 852 3002 1362
台湾 0800 666 052

facebook.com/redhatjapan
@redhatjapan
linkedin.com/company/red-hat

jp.redhat.com
#f8808_1017