E ブック

コンテナ化アプリケーションへの移行におけるベストプラクティス

背景
- 考え方
- 適切なレイヤー化
- 起動スクリプト
- 再起動について
- オーケストレーションの強化

アプリケーション要件
- アーキテクチャ
- セキュリティ
- パフォーマンス

テクニカル・チェックリスト
- ベストプラクティス
- アーキテクチャ
- セキュリティ
- パフォーマンス

まとめ

背景

アプリケーションのコンテナ移行における 3 つの主な戦略として、リフト & シフト、拡張、および書き換えが存在します。[1]

どの方式を採用する場合も、ほとんどのソフトウェアは最新のイメージベースのコンテナが発明されるよりも前に設計および作成されたものであると認識することが重要です。[2] 単一のコンテナ内でモノリシックなアプリケーションを実行する「リフト & シフト」方式を採用した場合であっても、多くの場合はアプリケーションに変更を加える必要があります。コンテナへの移行を成功させるには、アプリケーションの要件と Linux® コンテナの性質を考慮に入れた、堅固な移行戦略が必要です。

本書では、ソフトウェアをコンテナに移行する作業に関して、イメージのビルド手順から本番稼働における理想の状態まで、技術面における具体的な推奨事項やガイドラインを紹介します。アプリケーションの移行方法は、アプリケーションの要件によって決まります。

考え方

コンテナを使用することには、技術的なメリットに加えて業務上のメリットもあります。コンテナをビルドして使用するにあたっては、レイヤー化が重要になります。そのため、アプリケーションを分析し、どのような構成要素がどのように連携するかを検討する必要があります。この作業は、1 つのプログラムをクラスや関数へと分割していく作業と似ています。コンテナを構成するパッケージやスクリプトは、他のコンテナと連携し、アプリケーションをビルドします。つまり、アプリケーションは複数の小さな単位で構成されており、それらの単位をパッケージ化して扱いやすくするのがコンテナです。このように考えることで、コンテナ化アプリケーションの構造が理解しやすくなり、デプロイと保守も容易になります。

適切なレイヤー化

レイヤー化の目的は、前のレイヤーの上に薄い抽象化層を設けることで、より複雑なものを構築できるようにすることです。レイヤーは、同じタイプのオブジェクトをコンテンツとして有する、またはより小さな作業を実行する論理単位です。

レイヤーの数が適切であれば、コンテナは扱いやすくなります。レイヤー数が多すぎると過度に複雑化し、扱いが難しくなります。通常、適切なアプリケーションのレイヤー数は、アプリケーションの複雑さを反映します。つまり、アプリケーションが複雑であるほど、レイヤー数は多くなります。たとえば、Hello World というコンテナが標準出力 (stdout) に「Hello World」と出力する場合、構成、プロセス管理、および依存関係は不要なため、必要なレイヤー数は 1 層となります。しかし、Hello World アプリケーションを拡張し、ユーザーに対して「hello」と発信したい場合、入力を収集するための第 2 のレイヤーが必要になります。

1 “Containers for Grownups: Migrating Traditional & Existing Applications.” http://schd.ws/hosted_files/lcccna2016/91/Containers%20for%20Grownups_%20Migrating%20Traditional%20%26%20Existing%20Applications.pdf. アクセス日：2017 年 7 月 24 日。

2 “Containers for Grownups: Migrating Traditional & Existing Applications.” http://schd.ws/hosted_files/lcccna2016/91/Containers%20for%20Grownups_%20Migrating%20Traditional%20%26%20Existing%20Applications.pdf. アクセス日：2017 年 7 月 24 日。

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

jp.redhat.com

起動スクリプト

起動スクリプトを使用すると、コンテナ起動時に実行される薄い抽象化レイヤーを、効率的にサービスレイヤーの上に配置できます。起動スクリプトは、オペレーターがプラグインとして利用できる極めてシンプルなアプリケーション・プログラミング・インタフェース (API) を使用して、基本レイヤーの上に付加的な機能を提供します。起動スクリプトが行う最も一般的なタスクとして、アクセス許可の設定、設定ファイルの移動、ファイル所有権の変更、ディレクトリの消去、およびサービスの開始が挙げられます。

再起動について

再起動は、操作が実行されていることをプロセスに知らせる経済的かつ効率的な手段です。そのため、アプリケーションのあらゆるライフサイクル・アクションで再起動が使用されます。また、ライフサイクル操作では、プロセスの実行方法を変更するために再起動が必要とされます。

一例として、MariaDB に対する再構成操作の実行について考えてみましょう。設定ファイルの場所と起動情報をマッピングするシンプルなリモート・プロシージャ・コール (RPC) である JSON ファイルがあるとします。この JSON ファイルは set_configs.py によって解釈され、コンテナの起動時にその設定ファイルがそれぞれの場所にコピーされます。ユーザーは、ホスト上で構成設定を変更してコンテナを再起動することにより、MariaDB を再構成します。

オーケストレーションの強化

アプリケーションのレイヤーすべてがコンテナ内の配置に適しているわけではありません。そのため、追加するレイヤーが多すぎないようにしてください。レイヤーが多すぎると、既存のツールや他のレイヤーとすぐに重複してしまい、コンテナが過度に複雑化します。たとえば、アプリケーションのデプロイと管理のみ行えるツールをコンテナ内に構築するというのは、望ましくない余分な作業です。それよりも、適切にパッケージ化されたコンテナを、オーケストレーション・ツールを使用して簡単に管理できるようにするべきです。

アプリケーション要件

アプリケーションには、アーキテクチャ、セキュリティ、およびパフォーマンスの観点から、それぞれ固有の要件があります。一部の要件は、アプリケーションをコンテナに移行したり、複数のコンテナに分割したりするために必要とされる労力の程度に影響します。[3]

図 1. アプリケーション要件

アーキテクチャ

アーキテクチャの観点からすると、アプリケーションのコンテナ移行は、Unix から Linux への移行やオペレーティング・システムのアップグレードとさほど変わりません。多くの場合、アプリケーションは何年にもわたり稼働しています。アプリケーションに関する資料が存在しなかったり、情報が古くなっていたりすることも珍しくありません。多くの移行作業と同様に、移行作業を担当する技術者は、アプリケーションが構造的にどのように動作するのかを理解する必要があります。そのため、リバース・エンジニアリングを行い、アプリケーションがどのように作られているのかを調べる必要があります。そして、少なくとも以下のような情報が必要となります。

1. アプリケーションのバイナリーがどこに存在するか：インストーラーによってインストールされて 1 箇所に配置されているのか、それともファイルシステム全体に分散されているのか。簡単に起動できる単一のバイナリーが存在するのか、または使用できる単純な systemd ユニットファイルが存在するか。
2. アプリケーションのデータがどこにあるか：データは読み取り専用か、読み書き可能か。2 つの並行プロセスによる書き込みが安全に実施可能か。
3. すべての設定データがどこにあるか：単一のディレクトリ内や単一のファイル内、あるいはファイルシステム内の複数の場所に格納されているのか。
4. アプリケーションにどのような機密データが含まれているか：機密情報の場所は、アプリケーション内で設定可能か。情報は別個のディレクトリに移動可能か、あるいは何らかのキーを使用して ID または認証サーバー経由でアクセス可能か。
5. アプリケーションが必要とするネットワークアクセスの詳細： 単純な HTTP なのか。ユーザー・データグラム・プロトコル (UDP) を必要とするネームサーバーなのか。それとも、コンテナ間でポイントツーポイントの暗号化が必要で、インターネット・プロトコル・セキュリティ (IPsec) などを使用する、極めて複雑なアプリケーションなのか。
6. インストーラーの詳細：インストーラーは、リバース・エンジニアリングを行うことでアプリケーションのセットアップについての詳細が得られるようなシェルスクリプトなのか。バイナリーは、RPM またはその他の種類のパッケージ・マネージャーによってインストールされるのか。
7. アプリケーションのライセンス：コンテナイメージ内でアプリケーションの配信を容易に行えるようなものであるか。ライセンスは、極めて限定的な場合も、サイトライセンスが存在する場合もある。
8. アプリケーションの再起動が容易か：Apache はエラーなしに数千回再起動することが可能だが、データベースのテーブルは破損する可能性がある。これによりオーケストレーションと回復が困難になる可能性があるか。

3 「Container Tidbits: When Should I Break My Application into Multiple ....」2016 年 3 月 16 日、http://rhelblog.redhat.com/2016/03/16/container-tidbits-when-should-i-break-my-application-into-multiple-containers/。アクセス日：2017 年 3 月 22 日。

これらの質問に答えることにより、アプリケーションがコンテナ移行に適しているかどうかを判断できます。難易度が高すぎる場合、投資に見合った利益は得られません。[4]

表 1. データセンターにおける一般的なワークロード

	容易	中程度	困難
コーディング	完全分離 (単一プロセス)	ある程度分離 (複数プロセス)	自己変更 (アクターモデルなど)
設定	1 ファイル	複数ファイル	ファイルシステム内の どこか
データ	単一の場所に保存	複数の場所に保存	ファイルシステム内の どこか
機密情報	静的ファイル	ネットワーク	証明書の動的生成
ネットワーク	HTTP、HTTPs	TCP、UDP	IPSEC、高度に分離
インストール	パッケージ、ソース	インストーラーおよび 解明された設定	インストーラー (install.sh)
ライセンシング	オープンソース	プロプライエタリー	限定的、プロプライエタリー

セキュリティ

多くの点で、コンテナ化アプリケーションに関するセキュリティ上の決定は、プロセス内で実行される通常のアプリケーションと変わりません。[5] そのアプリケーションに適した分離度を決定する必要があります。移行へ向けてワークロードを評価し、コンテナによる分離が十分であるかどうかを判別する際に重要なのは、現在実行中の場所におけるワークロードの分離度がどの程度であるかを調べることです。

図 2 では、左から右に進むにつれて、テクノロジーの提供する分離度が高くなります。アプリケーションによっては、通常の Linux プロセスで提供される分離で十分です。MySQL と Web サービスを同一の Linux オペレーティング・システム・インスタンス上で実行することは一般的です。その一方で、同じアプリケーションの 2 つのコピーを、気候や地震のパターンが異なる、2 つの地域のデータセンターに配置しておかなければならない場合もあります。これは障害復旧のために一般的な措置です。

たとえば、現在、高パフォーマンス・コンピューティング (HPC) の複数のワークロードを、通常の Linux プロセスの分離のみが提供される大規模クラスタで実行することは一般的です。この場合、大規模クラスタの研究者が互いのプロセスのハッキングを試みることが可能であり、完璧とは言えません。しかし、通常その程度のリスクは許容範囲とみなされています。

もう 1 つの例として、仮想マシンにおいても、Oracle データベースや SAP インスタンスなどの内部向けサービスとはまったく異なる仮想化クラスタ内の外部ネットワークで、ドメイン・ネーム・サーバー (DNS)、HTTP、または仮想プライベート・ネットワーク (VPN) サービスなどの外部向けサービスを実行していることがよくあります。この設定は、ラックレベルの分離と同等になります。これら 2 種類のワークロードを同じ仮想化クラスタ内で実行することに賛同するセキュリティの専門家はいないでしょう。同様のことはコンテナ・プラットフォームについても言えます。これらのタイプのサービスは、別個のクラスタで実行するのが一般的です。

4 「Architecting Containers Part 4: Workload Characteristics and ....」2016 年 4 月 21 日、http://rhelblog.redhat.com/2016/04/21/architecting-containers-part-4-workload-characteristics-and-candidates-for-containerization/。アクセス日：2017 年 7 月 24 日。

5 「Why containers are really just fancy files and fancy processes.」2017 年 3 月 6 日、http://sdtimes.com/guest-view-containers-really-just-fancy-files-fancy-processes/。アクセス日：2017 年 3 月 22 日。

そのため、コンテナの提供する分離レベルで十分かどうか検討する際には、ワークロード要件の観点から考えましょう。

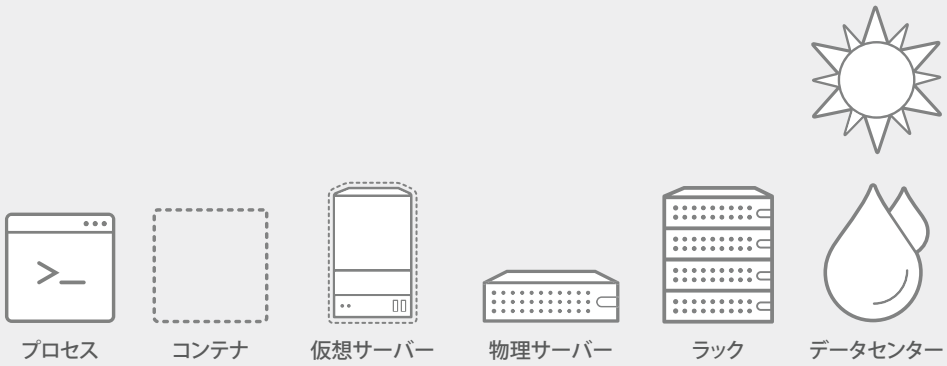

図 2. テナンシースケール

パフォーマンス

ワークロードのパフォーマンスについても分析する必要があります。コンテナと仮想化は付加的なテクノロジーであり、ベアメタル・ハードウェアと組み合わせることによって、さまざまな機能を提供します。表 2 は、アプリケーションをコンテナへ移行する際に考慮すべき重要な機能や能力を見極めるためのクイックガイドです。

コンテナとは Linux プロセスの一種であり、コントロールグループ (cgroups)、Security-Enhanced Linux (SELinux)、および名前空間などのテクノロジーを使用することで、より高度な分離をアプリケーションへ提供します。これにより、ネイティブ動作時と同様か、それに近いスピードでアプリケーションを実行できるようになります。仮想化のような抽象化レイヤーはないため、クラスタ内のコンテナ化アプリケーションは、すべて同一のハードウェア・アーキテクチャおよびオペレーティング・システムに基づくものでなければなりません。

前述の例の場合、HPC 環境においてコンテナをベアメタルに追加することにより、同レベルのパフォーマンスを実現しながら分離度を上げることが可能です。一方、ワークロードが Windows と Linux に配置されたコンポーネントを必要とする企業アプリケーションである場合、コンテナと仮想化を組み合わせることが推奨されます。コンテナを仮想マシン内で実行すると、ハードウェアの自由度、分離度の向上、コンテナイメージを使用した管理可能性の向上が、すべて実現します。[6]

6 「When Containers and Virtualization Do - and Don't - Work Together ....」https://summits.brighttalk.com/webinar/when-containers-and-virtualization-do-and-dont-work-together/。アクセス日：2017 年 3 月 22 日。

表 2 は、複数のテクノロジーを組み合わせることのメリットとデメリットを表しています。

表 2. ワークロード・プラットフォームの比較

	ベアメタル	+ コンテナ	+ 仮想化
CPU 高負担	高速	高速	高速
メモリー高負担	高速	高速	高速
ディスク入出力レイテンシー	高速	高速	普通
ディスク入出力スループット	高速	高速	高速
ネットワーク・レイテンシー	高速	高速	普通
ネットワーク・スループット	高速	高速	高速
デプロイ速度	低速	高速	普通
アップタイム (ライブ・マイグレーション)	×	×	○
代替 OS	○	一部	○

テクニカル・チェックリスト

ベストプラクティス

- アプリケーションをレイヤー化する
- レイヤーは、アプリケーションの複雑さに応じた数を設ける
- コンテナは、RPM よりも若干高度な抽象化であるようにする
- すべての問題をコンテナ内で解決しようとしない
- 起動スクリプトレイヤーを使用して、プロセスランタイムからシンプルな抽出を行う
- 外部ツールから制御するコンテナ内に明確で簡潔なオペレーションを組み込む
- コード、設定、データを識別し、分離する
- コードはイメージレイヤー内に配置する
- 設定、データ、および機密情報は、環境から取得する
- コンテナは再起動に対応している
- プロセスを再作成しない
- latest タグのついたイメージをベースに作業しない (長期的にビルドの再現性が損なわれるため)
- Liveness Probe と Readiness Probe で動作と通信可能性のヘルスチェックを行う

アーキテクチャ

このチェックリストは、コンテナ化アプリケーションのイメージを作成する際に使用します。[7] 以下の情報を把握しておくことが重要です。

1. アプリケーションをコンテナ内で実行するために必要なすべてのバイナリーの場所を特定する
   a. レイヤーを使用する：この際、コアビルドとアプリケーション・ランタイム・レイヤーについても検討します。[8,9]
   b. 依存関係を特定する：それらの依存関係を前のレイヤーに含めるべきかどうか、特にそれらを他のアプリケーションで共有または使用できるかどうかも見極めます。
   c. バイナリーの起動方法 (スクリプト、systemd など) を特定する
2. アプリケーションの設定情報を含むファイルとディレクトリを特定する：これらは、実行時にアプリケーションにバインドマウントする必要があります。設定は環境 (開発/QA/プロダクション) から取得し、コンテナイメージの内部に埋め込まないようにします。
3. アプリケーションのデータを含むファイルとディレクトリを特定する：これらは、実行時にアプリケーションにバインドマウントする必要があります。アプリケーションのデータは環境 (開発/QA/プロダクション) から取得し、コンテナイメージの内部に埋め込まないようにします。
4. アプリケーションが必要とするネットワーク・プロトコルを特定する：これにより、これらのサービス[10]が内部のクラスタに提供されるか、または外部のカスタマーに提供されるかが決定します。
5. インストーラー・スクリプトをリバース・エンジニアリングすることによって、動作方法の詳細を確認することが可能かどうか
   a. スクリプトがどのような設定変更を適用するのか特定する：インストーラー・スクリプトの代わりに、スクリプティング管理や設定管理の機能を使用できるかどうか確認します。また、実行時に設定をコンテナイメージに渡して、パラメーターを動的に設定することが可能かどうか確認します。
   b. データの格納場所を特定する：インストール中にデータのスキーマがセットアップされているかどうか、また、それをアプリケーション実行時にスクリプティングできるかどうかを調べます。
6. サービスの再起動が容易かどうかを確認する：サービスが再起動の影響を受ける場合、Liveness Probe と Readiness Probe を使用して動作と通信可能性のチェックを行い[11]、オペレーターの介入が必要かどうかを調べます。コンテナ・オーケストレーション環境では、可能な限り多くのことを自動化することが重要です。
7. ロギングと出力の宛先を特定する：通常、RPM でインストールされるアプリケーションのログは /var/log またはその他の既知の場所に入ります。コンテナ化環境では、これにより読み書きレイヤーに大量のデータがダンプ出力される可能性があります。[12]

---

7 「GitHub - opencontainers/image-spec: OCI Image Format.」https://github.com/opencontainers/image-spec。アクセス日：2017 年 3 月 22 日。

8 「Architecting Containers Part 5 - Red Hat Enterprise Linux Blog.」2016 年 5 月18 日、http://rhelblog.redhat.com/2016/05/18/architecting-containers-part-5-building-a-secure-and-manageable-container-software-supply-chain/。アクセス日：2017 年 3 月 22 日。

9 「GitHub - fatherlinux/container-supply-chain: Demo showing how to ....」https://github.com/fatherlinux/container-supply-chain. アクセス日：2017 年 3 月 22 日。

10 「Services - Kubernetes.」http://kubernetes.io/docs/user-guide/services/. アクセス日：2017 年 3 月 22 日。

11 「Configuring Liveness and Readiness Probes - Kubernetes.」http://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/。アクセス日：2017 年 3 月 22 日。

12 「GitHub - fatherlinux/container-internals-lab: Container internals lab for ....」https://github.com/fatherlinux/container-internals-lab。アクセス日：2017 年 7 月 17 日。

8. 個々のプロセスのスケーリングが必要かどうかを判別する：必要に応じてアプリケーションを複数のコンテナに分割します。[13]

セキュリティ

このチェックリストは、コンテナ化アプリケーションのイメージを作成する際に使用します。[14] 以下の情報を把握しておくことが重要です。

1. 可能な場合は必ず、ポリシーを使用する：セキュリティ・コンテキストの制約およびサービスアカウント[15]を使用して、それらのポリシーを設定します。Red Hat の調査によると、アプリケーションごとにカスタムルールを定めるよりも、プロファイルベースでコントロールする方法のほうが優れており、一般により幅広く採用され、保守されています。
2. アプリケーションの機密情報[16]を含むファイルを特定する：このファイルには、証明書やパスワードなどが入っています。このような機密情報は、決してコンテナイメージに埋め込まないでください。埋め込んだ場合、イメージをダウンロードすればだれでも機密情報にアクセスできてしまうためです。機密情報はコンテナ環境によって保護し、提供する必要があります。
3. 特権を必要とする低ポート上でコンテナを実行しないようにする：これを防ぐため、組み込みセキュリティ・コンテキストの制約[17]を使用します。
4. コンテナを root として実行しないようにする[18]：カーネルのユーザー名前空間を使用していたとしても、コンテナ外に権限昇格するリスクがあり、結果として基盤となるコンテナホストを危険にさらす可能性があります。これを防ぐため、ビルトインのセキュリティ・コンテキストによる制約を使用します。
5. 可能な場合、コンテナを読み取り専用の root ファイルシステムで実行する：これには、セキュリティ・コンテキストによる制約を使用します。
   a. Web コンテンツの場合：読み取り専用ボリュームマウントでもサービスを実行します。[19]
6. 必要な分離度を特定する：最低特権[20]および多層防御の原則を使用します。[21] Red Hat® OpenShift Container Platform 環境では、セキュリティ・コンテキストの制約とサービスアカウントにより、以下のテクノロジーを設定します。

13 「Container Tidbits: When Should I Break My Application into Multiple ....」2016 年 3 月16 日、http://rhelblog.redhat.com/2016/03/16/container-tidbits-when-should-i-break-my-application-into-multiple-containers/。アクセス日：2017 年 7 月 24 日。

14 「GitHub - opencontainers/image-spec: OCI Image Format.」https://github.com/opencontainers/image-spec。アクセス日：2017 年 3 月 22 日。

15 「Understanding Service Accounts and SCCs – OpenShift Blog.」2016 年 4 月 15 日、https://blog.openshift.com/understanding-service-accounts-sccs/。アクセス日：2017 年 7 月 24 日。

16 「Secrets - Kubernetes.」http://kubernetes.io/docs/user-guide/secrets/。アクセス日：2017 年 3 月 22 日。

17 「Authorization - Additional Concepts | Architecture | OpenShift ....」https://docs.openshift.com/container-platform/3.5/architecture/additional_concepts/authorization.html。アクセス日：2017 年 7 月 21 日。

18 「Can I haz non-privileged containers? by mhausenblas.」http://canihaznonprivilegedcontainers.info/。アクセス日：2017 年 7 月 24 日。

19 「Volumes | Kubernetes.」https://kubernetes.io/docs/concepts/storage/volumes/。アクセス日：2017 年 7 月 24 日。

20 「What is principle of least privilege (POLP)? - Definition from WhatIs.com.」http://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP。アクセス日：2017 年 7 月 21 日。

21 「GitHub - fatherlinux/container-defense-in-depth.」https://github.com/fatherlinux/container-defense-in-depth。アクセス日：2017 年 7 月 21 日。

a. SELinux：Red Hat Enterprise Linux システムには、そのまま使える設定可能なプロファイルが付属しています。[22] コンテナごとに、セキュリティ保護された仮想化 (sVirt) を使用して複数の動的コンテキストが生成されます。[23]

b. セキュア・コンピューティング (seccomp)[24]：コンテナは、デフォルトの seccomp プロファイルなしで実行されます。ユーザー自身がプロファイルを特定し、設定する必要があります。

c. Linux の機能：[25]

   i. OpenShift 上のデフォルトのセキュリティ・コンテキストによる制約は限定されており、KILL、MKNOD、SYS_CHROOT、SETUID、SETGID の機能は含まれていません。

   ii. 管理者は、特に root 特権（管理者にとって有用）が付与されるプロセスの場合、次の機能を制限するカスタム制約を作成できます：AUDIT_CONTROL、BLOCK_SUSPEND、DAC_READ_SEARCH、IPC_LOCK、IPC_OWNER、LEASE、LINUX_IMMUTABLE、MAC_OVERRIDE、および MAC_ADMIN。

   iii. 管理者は、特に root 特権（管理者の作業にとって有用）が付与されるプロセスの場合、次の機能を付与するカスタム制約を作成できます：NET_ADMIN、NET_BROADCAST、SYS_ADMIN、SYS_BOOT、SYS_MODULE、SYS_NICE、SYS_PTRACE、SYS_PACCT、SYS_RAWIO、SYS_RESOURCE、SYS_TIME、SYS_TTY_CONFIG、SYSLOG、および WAKE_ALARM。

d. セキュリティ・コンテキストによる制約：Red Hat OpenShift Container プラットフォームによって提供され、優れたデフォルトルールとなります。

パフォーマンス

このチェックリストは、コンテナ化アプリケーションのイメージを作成する際に使用します。[26] 以下の情報を把握しておくことが重要です。

1. アプリケーションが一時ファイルにアクセスしたり一時ファイルを作成したりするかどうかを判別する：デフォルトの場合、それらのファイルシステムは読み取り専用でマウントされます。これらのファイルシステムをボリュームとしてマウントすることは可能ですが[27]、複数プロセスからの読み書きアクセスは慎重に行ってください。

   a. /sys

   b. /proc:

      i. カーネルデータ構造へのインタフェースを提供する擬似ファイルシステムです。

      ii. ほとんどは読み取り専用ですが、一部のファイルではカーネル変数を変更できます。

22 「第6章 Docker SELinux セキュリティポリシー - Red Hat Customer Portal」 https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy。アクセス日：2017 年 7 月 21 日。

23 「Securing Docker Containers with sVirt and Trusted Sources - Crunch ....」 2015 年 5 月 21 日、http://crunchtools.com/securing-docker-svirt/。アクセス日：2017 年 7 月 24 日。

24 「Authorization - Additional Concepts | Architecture | OpenShift Origin ....」 https://docs.openshift.org/latest/architecture/additional_concepts/authorization.html。アクセス日：2017 年 7 月 24 日。

25 「Capabilities(7) - Linux manual page - man7.org.」 http://man7.org/linux/man-pages/man7/capabilities.7.html。アクセス日：2017 年 7 月 24 日。

26 「GitHub - opencontainers/image-spec: OCI Image Format.」 https://github.com/opencontainers/image-spec。アクセス日：2017 年 3 月 22 日。

27 「Volumes | Kubernetes.」 https://kubernetes.io/docs/concepts/storage/volumes/。アクセス日：2017 年 7 月 24 日。

iii. /proc/[pid]ns 内のエントリーは、コンテナ化プロセス ID に対してインスタンス生成されるカーネルの名前空間を表します。[28]

iv. 例：/proc/asound、/proc/bus、/proc/fs、/proc/irq、/proc/sys、および /proc/sysrq-trigger。

c. /dev:

i. コンテナ内部で、アプリケーションは /dev/null や /dev/zero など、限定されたデバイスファイルにアクセスできます。コンテナ化アプリケーションは、特権モードを使用せずに /dev/sdX や /dev/ttySX などのホストデバイスにアクセスすることはできません。

d. /run:

i. docker v1.10 以降では、docker run に tmpfs オプションを渡すことができます。そうすると、/run はコンテナ内に tmpfs としてマウントされます。

ii. Red Hat システムでは、/run/secrets は常に tmpfs としてマウントされ、サブスクリプション情報の格納場所になります。

iii. Linux ホストでは、/run は mpfs としてマウントされ、プロセスの一時データ (デーモンの pid など) がそこに保存されます。この一時データはサーバーを再起動すると削除されます。コンテナ内では、/run/secrets のみ tmpfs としてマウントされます。/run 自体は / (root) ファイルシステムに内包されます。したがって、/run 以下のファイルはコンテナを再起動しても削除されません。

2. アプリケーションが、カーネル・パラメーター (/proc/sys) に変更を加えたり、特殊ハードウェアにアクセスしたりする必要があるかどうかを判別する：

a. デフォルトの場合、/proc/sys の以下にあるカーネルのチューニングを行う変数は、コンテナ化プロセスでは読み取り専用となります。

b. これらのタイプのアプリケーションを実行するには、特定のノードを適切なカーネルパラメーターやハードウェアにより設定し、ノードセレクターを使用して特殊な設定やリソースを持つノード上でアプリケーションをスケジューリングしなければならない場合があります。[29] 例：/proc/sys/fs/mqueue、 /proc/sys/kernel/{msgmax, msgmnb, msgmni, sem, shmall, shmmax, shmmni, and shm_rmid_forced}

c. アプリケーションでカーネルパラメーター自体に (sysctl などを使用して) 変更を加えることが必要な場合、その変更操作は、特権コンテナを含む隔離されたクラスタで実行する必要があります。

3. 日付、時刻、およびロケール：

a. アプリケーションで異なるロケール設定 (JST など) が必要かどうかを判別します。このロケール設定は、ビルド時のイメージで行います 。そのため、イメージを再ビルドします。[30]

b. アプリケーションで日付を変更する必要があるかどうかを判別します。[31]

---

28 「namespaces(7) - Linux manual page - man7.org.」 http://man7.org/linux/man-pages/man7/namespaces.7.html。アクセス日：2017 年 7 月 24 日。

29 「Simple use of selectors to get pods to land on the desired nodes ....」 2016 年 8 月 12 日、https://blog.openshift.com/use-of-selectors-to-get-pods-on-desired-nodes/。アクセス日：2017 年 7 月 24 日。

30 「Changing The Time Zone In Linux (Command Line) - Linux Academy ....」 2012 年 7 月 30 日、https://linuxacademy.com/blog/linux/changing-the-time-zone-in-linux-command-line/。アクセス日：2017 年 7 月 24 日。

31 「java - Is it possible change date in docker container? - Stack Overflow.」 2015 年 4 月 10 日、http://stackoverflow.com/questions/29556879/is-it-possible-change-date-in-docker-container。アクセス日：2017 年 3 月 22 日。

4. アプリケーションが、固定 IP アドレスの使用を想定しているかどうかを判別します。
   a. 可能であれば静的 IP の設定は避けます。
   b. コンテナ・ネットワーク・インタフェースの IP アドレスは変更できません。
   c. ネットワーキングは Kubernetes のサービスネットワークが処理を行うため、可能であればホスト名を使用します。
   d. アプリケーションに IP アドレスを含むパラメーターまたは設定がある場合、ENTRYPOINT をインターセプトして、起動時に設定ファイルを動的に変更します。この操作は、SED や Ansible® などのツールを使用して実行します。
5. アプリケーションで、複数のネットワーク・インタフェースの使用が必要かどうかを判別します。
   a. Kubernetes では、現在のところ複数の仮想ネットワーク・インタフェース・コントローラー (NIC) はサポートされていません。コンテナ内では、ボンディングなどのネットワーク冗長機能は使用できません。この機能はホストレベルで実行する必要があります。Pod を設計する際は、障害が発生した場合に作動中のネットワークのノード上で再起動するようにします。適切な Liveness と Readiness のチェックを設定します。
   b. サードパーティ製ツールを使用することで、複数ネットワークのインタフェースを実現可能です。[32]

まとめ

既存のアプリケーションを 1 つまたは複数のコンテナに移行する作業を成功させるには、アプリケーションについてよく理解し、総合的な計画を立てることが必要です。ほとんどのアプリケーションはコンテナ化できますが、必要な作業量についてよく理解し、コンテナに移行してもパフォーマンスが維持され、セキュリティが低下しないようにしましょう。

32 「Support multiple pod IP addresses · Issue #27398 · kubernetes ....」2016 年 6 月 14 日、https://github.com/kubernetes/kubernetes/issues/27398。アクセス日：2017 年 7 月 24 日。