Red Hat Enterprise Linux Atomic Host 7

コンテナー開発の推奨プラクティス

コンテナー開発のベストプラクティス

Last Updated: 2018-02-07

Red Hat Enterprise Linux Atomic Host 7 コンテナー開発の推奨プラクティス

コンテナー開発のベストプラクティス

法律上の通知

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

長期的にサポートされる有効なコンテナー関連の基本的な手法についての一般的な推奨事項です。

目次

第1章 概要

本書は、コンテナー開発の推奨プラクティスガイドです。本書は、ソリューションアーキテクトによる Red Hat がサポートするコンテナー開発の一連のガイドラインのリクエストに対応して作成されました。現在の Docker ベースのインプリメンテーションにおけるコンテナーは急速に進歩している新テクノロジーですが、本書の作成にあたって、Red Hat 内のコンテナーのサポート状況を確認し、これを分かりやすい方法で説明することを試みました。コンテナーの使用事例は多数あるため、ここでは長期的にサポートされる有効なコンテナー関連の基本的な手法について一般的な推奨を行います。

1.1. コンテナーの出所

ここでは、コンテナーの出所、コンテナーの取得方法、かつコンテナーを安全に取得する方法について説明します。

コンテナーはイメージから構築され、イメージはレジストリーにあるリポジトリーに保存されます。本書では、コマンド **docker pull** がアクセスできる、(1) docker.io レジストリーおよび (2) 内部 redhat.com レジストリーの 2 つのレジストリーについて説明します。

1.1.1. 「docker pull」の危険性

素 (naked) の状態で使用される **docker pull** は危険なコマンドになる可能性があります。Docker はソフトウェアの取得とソフトウェアのインストールをほとんど区別しません。これは (たとえば) RPM の場合と異なります。RPM では、これをインストールしない限り、**wget** を使用してマルウェアを含む RPM を取得しても問題はありません。ただし、Docker を使用してマルウェアを取得するのは危険です。取得の動作はインストールの動作と機能的に同等です。

コンテナーを、取得時に特権付きで開始されるソフトウェアであると考えてください。コンテナーでは設定の操作が行われた後にのみ特権が解除されます。複数コンテナーの分離は通常は任意で行われ、デフォルトでは分離されません。

docker pull コマンドを使用する際は注意してください。

また、**docker pull** コマンドの使用に危険が伴う理由を理解しておく必要があります。

コンテナーが RH レジストリーにない場合、**docker pull** は docker.io レジストリーにフェールオーバーします。Red Hat は **docker.io** から派生するコンテナーのセキュリティーまたは信頼性を検証しません (Docker Inc. も同様です)。イメージの名前付け規則のセクションも参照してください。

Red Hat レジストリー以外の場所からイメージを取得する場合、**docker pull** の使用を避けてください。可能な場合は、 **docker load** および **docker save** を使用します。**docker load** および **docker save** を tarball で使用し、それらを Fedora 内で確認することができます。

docker load および **docker save** を **docker pull** の代わりに使用する必要がある理由として、**docker load** および **docker save** の使用は、**docker pull** を実行する場合に Docker Hub へのシステム公開時に避けられないセキュリティーの脆弱性を防ぐ手段となります。

適切なコンテナーの場所および **docker pull** を使用する際に注意する点については、2014 年 12 月 18 日付の Trevor Jay 氏による Red Hat Security Blog の掲載記事 Before You Initiate a "docker pull" を参照してください。

1.1.1.1. Docker Pull

docker pull の基本的な形式は **$ sudo docker pull repo/image:tag** です。ここで、**repo** および **tag** はオプションになります。**repo** および **tag** が指定されない場合、docker はイメージを

docker.io レジストリー内で見つけようとします。このため、イメージを取得するレジストリーの名前を常に明示的に設定することが推奨されます。レジストリーが指定されない場合も、docker は docker.io レジストリーでイメージを見つけようとします。tag が指定されていない場合、docker はデフォルトで最新イメージを取得しようとします。

1.1.1.2. Docker Load

docker load の基本的な形式は以下のようになります。

```
$ sudo docker load -i input.tar
```

ここで、**input.tar** はローカルの docker レジストリーにロードされる tar イメージです。**-i** および **input.tar** ファイル名はオプションです。**-i** またはファイル名のいずれも指定されない場合、**docker load** は STDIN の tar データを予想します。

1.1.1.3. Docker Save

docker save の基本的な形式は以下のようになります。

```
$ sudo docker save -o output.tar
```

ここで、**output.tar** はローカルの docker レジストリーにロードされる tar イメージです。**-o** および **output.tar** ファイル名はオプションです。**-o** またはファイル名のいずれも指定されない場合、**docker save** は STDOUT にコンテナーデータを出力します。

第2章 コンテナーの構築

セキュリティーの脆弱性の検出時にコンテナーにパッチを適用できるよう、コンテナーのアップデートが可能なメカニズムを維持する必要があります。そのためには、コンテナーが構築される方法を理解しておく必要があります。

全 **docker** コンテナーの **30%** には **CVE** (セキュリティー脆弱性) が関連付けられています (**docker** コンテナーの脆弱性についての詳細は、http://www.banyanops.com/blog/analyzing-docker-hub/ を参照してください)。

これらの **CVE** に関連する脅威に効果的に対応するには、コンテナーにセキュリティーパッチを適用できるよう、コンテナーをアップデートできる状態にする必要があります。コンテナーを安全にアップデートできる状態にしておく唯一の方法は、既存のパッケージインフラストラクチャーとツールをできるだけ多く使用することにあります。つまり、**dnf (yum)** を使用する必要があります。これに関して、コンテナーを実行することは他のディストリビューションを実行することと全く変わりがありません。一般的なすべてのルールが適用されます。可能であれば **RPM** を使用する必要があります。

第3章 ラベル

ラベルを使用すると、ラベルが存在する以前に必要であった手間のかかる設定手順が不要になります。これらの設定手順には、コンテナーが特権コンテナーとして実行する必要があるかどうかを確認することも含まれました。

これまでは、この種の設定手順を把握した上でコンテナーごとに多大な労力を要する設定作業が必要でした。ただし、コンテナーを定義する **.json** ファイルの一部であるラベルにより、この情報が **docker** に提供されるようになりました。ラベルは、**docker** で利用可能な通常のメタデータに基づいて作成されます。

「ラベル」は **docker** メタデータに基づいて作成されるオープン標準です。**docker** には、名前と値のペアをコンテナーに追加できる機能があります。ラベルは、この名前と値のペアのコンテナーへの挿入を可能にする機能を使用して作成されます。

3.1. ラベルのサンプル

ラベルを使用すると、ゴール達成までにコマンドラインに入力しなければならないすべての詳細を把握しておく必要がなくなります。以下の例は、ラベルの性質と機能を理解するのに役立ちます。

ここでは、一定の意味を持つ「**run labels**」を作成します。

GPU アクセラレーションと **pulse audio** 搭載の **Chrome** を実行するとします。これを実行するには、コンテナーが GPU にアクセスできる必要があります。それ自体は難しいことではありませんが、「合成 (composition)」について理解する必要があります。これを説明するために、ここでは連携する **X11** と **Window Manager** の例を使用します。これらの **2** つのプログラムを最も効率よく連携させる方法として、合成要素のすべてのプログラムの書き込み先となる **1** つの大きなフレームバッファーを作成できます。これは、次のような非常に長いコマンドを使用して処理されます。

```
$ sudo docker run -v /dev/dri:/dev/dri \
-v /dev/snd:/dev/snd \
-v /dev/shm:/shm \
- ipc=container:foo_bar \
- privileged -e 'DISPLAY=:0' \
-u username fedora_chrome google-chrome
```

以下は内訳になります。

-v /dev/dri:/dev/dri

ビデオカードコンポーネント

-v /dev/snd:/dev/snd

サウンドカードコンポーネント

-v /dev/shm:/dev/shm

共有メモリー

shm

共有メモリー

- ipc

プロセス間通信

container:foo_bar は、このラベルが「**foo_bar**」と呼ばれる別の実行中のコンテナーを参照することを意味します。これは実行中のコンテナー **foo_bar** を実行中のコンテナー **fedora_chrome** にリンクします。この引数は **2** つのコンテナーのプロセス間通信の名前空間を共有し、**2** つのコンテナーが

同じオブジェクトを参照し、2 つのコンテナーのそれぞれがそれらのオブジェクトを参照するために同じ名前を使用するようにします。これは、それらの通信が **/dev/shm** で実行されるために重要となります。

共有メモリーはプロセス間通信に必要な 1 つの手段です。

-e 'DISPLAY=:0'

出力の移動先の X11 ディスプレイ

-u username

ユーザー名を指定します。

fedora_chrome

コンテナーの名前です。

google-chrome

コンテナーが実行するコマンドです。

第4章 イメージの名前付け規則

4.1. RED HAT の名前付けポリシー

Red Hat は、ユーザーの予測性を高めるために、一般にリリースされた Docker イメージについての一貫性のある名前付けポリシーを提供しています。

プロトコル/レジストリー形式の V1 バージョンの Docker URL は GitHub リポジトリー名と同様に機能します。以下のような構造になります。

```
REGISTRY[:PORT]/USER/REPO[:TAG]
```

暗示的なデフォルトレジストリーは **docker.io** です。これは、"redhat/rhel" などの相対 URL が **docker.io/redhat/rhel** に解決されることを意味します。

特殊名の "library/*" は、直接の接頭辞のないイメージにマップされます。例: "docker.io/rhel".

4.2. 要求のリダイレクト

Red Hat コンテンツの要求は **docker.io** から **registry.access.redhat.com** にリダイレクトされています。以下のマッピングはこのリダイレクトについて説明しています。

- docker.io/rhel → registry.access.redhat.com/rhel (alias for rhel7)
- docker.io/rhel7 → registry.access.redhat.com/rhel7
- docker.io/rhel6 → registry.access.redhat.com/rhel6
- docker.io/redhat/* → registry.access.redhat.com/redhat/*

Red Hat のコミュニティーのコンテンツには、以下の場所にある Docker Hub でアクセスできます。

- docker.io/fedora
- docker.io/centos
- docker.io/jboss/*

Red Hat Enterprise RHEL 7.2 に同梱される Docker のバージョンは、デフォルトで内部の Red Hat レジストリーと通信します。これにより、Red Hat は名前空間をレジストリーでセグメント化できます。

レジストリーコンテンツはタグ付けされた名前のマッピングであり、コンテンツのコピー操作を伴うものではありません。このマッピングは複数のシンボリックリンクを使って行われます。Red Hat はこれらのリンクを常に最新の状態に保つことでデータの整合性を確保します。

「REPO」は、多数の明示的にタグ付けされたイメージと多数の表示されない層が含まれるリポジトリーです。「IMAGE」はリポジトリー内の特定の層が設定されたブランチです。「イメージ」と「リポジトリー」という用語は同意語として使用されることが多くありますが、両者は同じものを指す訳ではありません。たとえば、複数の異なるイメージは単一リポジトリーにタグ付けすることができます。

第5章 プロセス管理

このセクションではプロセス管理について扱いますが、以下の 2 つの異なるケースについて考慮します。

1. 単一プロセスコンテナー: コンテナーのジョブの実行対象となるファイルを保管します。
2. 管理対象の複数プロセスコンテナー: 同時に実行される複数の異種プロセスから構成されます。

Apache は単一プロセスコンテナーの一例です。Apache はすべてのジャーナリングを独自に実行します。外部コンテナーがそのジャーナリングを実行する必要はありません。つまり、この場合 **systemd** を使用しても意味がないことになります。

GNOME3 は管理対象の複数プロセスコンテナーの一例です。この種のコンテナーを手動で管理する試み (たとえば、**upower** および **dbus** ならびに **logging** を連動させること) は、単純に **systemd** を機能させるよりも煩雑になります。

5.1. コンテナーで SYSTEMD を実行する

管理対象の複数プロセスコンテナーの場合に、コンテナーで実行される複数のプロセスを管理するために **systemd** を実装します。**systemd** をコンテナーで実行するには、以下のコマンドと同様のコマンドを実行します。

```
$ sudo docker run -it --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro
IMAGE /bin/bash
```

注記

上記のコマンドは、**systemd** を実行するインタラクティブな **-tty (-it)** コンテナーを起動します。Docker のバージョンによっては、**`--privileged`** フラグは不要になります。

コンテナーで **systemd** を実行することに関する詳細な背景情報およびコンテキスト情報については、この点について論じている Dan Walsh 氏の 2014 年 5 月のブログ記事を参照してください。Running systemd within a Docker Container

5.1.1. コンテナーでの journald および systemd

コンテナー内でジャーナル監査 (journal auditing) を無効にします。docker ホストのみが **journald** を実行できるように許可します。ジャーナル監査はカーネル機能であり、一度にこの機能を使用できるのは 1 つの journald のインスタンスのみです。(docker ホストが journald を実行中の場合にはこの限りではありません。)

5.1.2. サービスおよびコンテナー

サービスは、コンテナーでは追加の設定なしで機能します。サービスはコンテナーでは「機能するのみ」となります。つまり、サービスをコンテナーで機能させるために特殊な設定は必要ありません。

5.1.3. コンテナー外のハードウェアの管理

外部のハードウェアを管理するために **systemd** を使用するコンテナーを作成することができます。ただし、コンテナーを設定すると、管理用のコンテナーを設定する際に使用した特定のハードウェアセットアップが想定されるようになるため、コンテナーの移植性はなくなります。

5.1.4. systemd およびゾンビ状態

systemd はゾンビ状態の問題を解決します。プロセスが終了し、ゾンビになる場合、**init 1** はそれらのプロセスを回収 (reap) します。これはコンテナー外でもコンテナー内でも同様です。**systemd** がコンテナー内のゾンビプロセスを 回収 (reap) する方法は、コンテナー外にあるゾンビプロセスを回収 (reap) する方法と全く同じであることを予想できます。

systemd (または、**sysv**) なしにコンテナーを起動する場合、そのコンテナーで終了するプロセスの回収 (reap) が行われることはありません。

第6章 LINTER FOR DOCKERFILE

Linter for Dockerfile は、Dockerfile にエラーがあるかどうかを調べるユーティリティーです。

Linter for Dockerfile は、お客様およびパートナー様が有効な Dockerfile を構築するのに役立つユーティリティーです。Linter for Dockerfile は、パートナーテクノロジーを採用しているお客様およびパートナー企業の皆様が Red Hat がコンテナーをできる限り使いやすくするための調整に前向きに取り組んでいることをご理解いただくことを目的として開発されました。

6.1. LINTER FOR DOCKERFILE の使用方法

Red Hat ログインおよび有効なお客様またはパートナー様のエンタイトルメント (有効なサブスクリプション) を持つすべてのユーザーが Linter for Dockerfile にアクセスできます。

以下から Linter for Dockerfile ユーティリティーにアクセスします。https://access.redhat.com/labs/linterfordockerfile/#/

Dockerfile を検証するには:

1. お客様またはパートナー企業の皆様が Dockerfile をアップロードします。Dockerfile の URL を指定するか、または Dockerfile の内容を貼り付けるかのいずれかを実行します。
2. お客様またはパートナー企業の皆様が特定のプロファイル (例: "Red Hat ISV") を選択します。
3. お客様またはパートナー企業の皆様が「Check」を選択します。
4. Linter for Dockerfile は構文の色別表示で Dockerfile の内容を表示します。
5. Linter for Dockerfile は、一連のルールに対して Dockerfile の内容を比較します。
6. Linter for Dockerfile は構文エラーを表示し、お客様またはパートナー企業の皆様によって選択されたプロファイルに関連付けられたベストプラクティスについて説明したフィードバックを提供します。