Red Hat Enterprise Linux Atomic Host 7
コンテナーの使用ガイド

コンテナーの使用ガイド

Red Hat Atomic Host Documentation Team

コンテナーの使用ガイド

法律上の通知

概要

コンテナーおよびコンテナー開発

目次

第1章 LINUX コンテナーの紹介

1.1. 概要

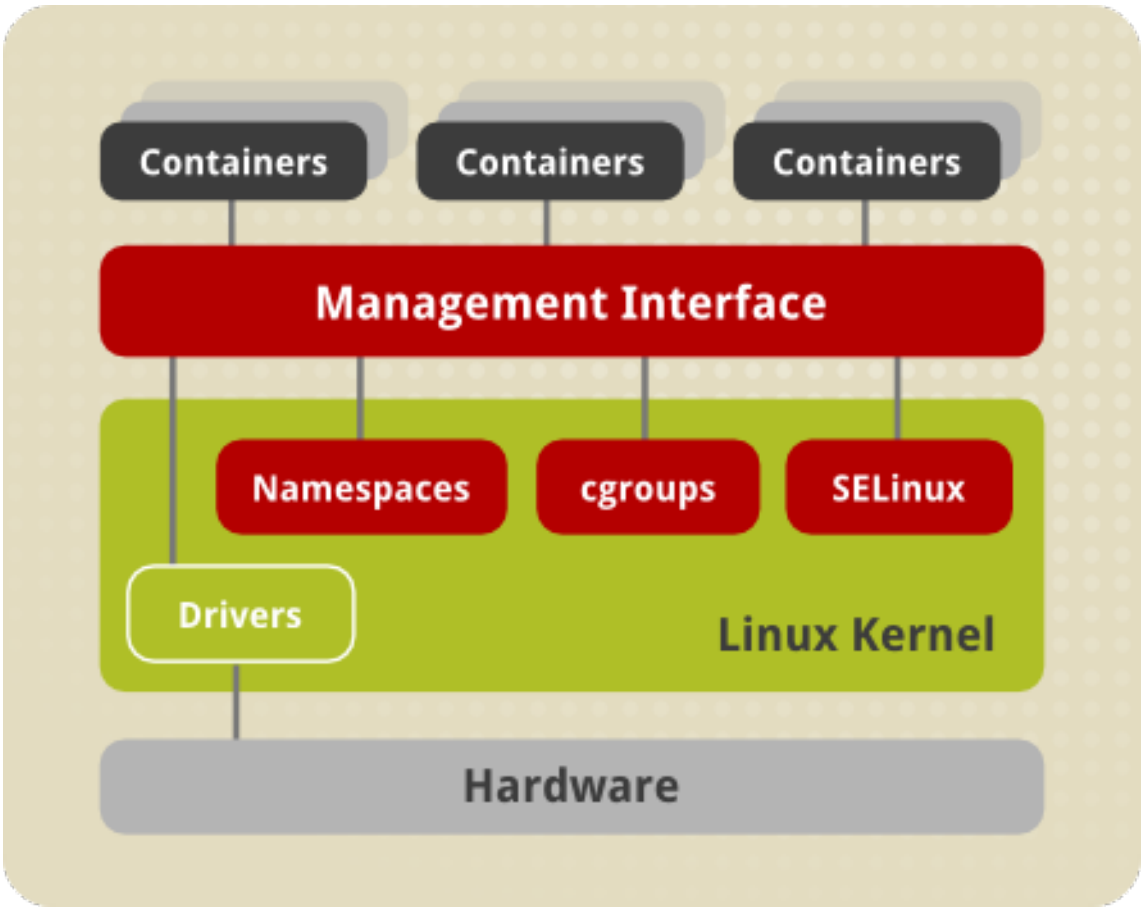
Linux コンテナーは、軽量なアプリケーションの分離と柔軟なイメージベースのデプロイ方法を組み合わせた、主要オープンソースアプリケーションのパッケージングおよび配信テクノロジーとして登場しました。

Red Hat Enterprise Linux 7 では、リソースを管理するコントロールグループ (cgroup)、プロセスを分離する名前空間、セキュリティーを担う SELinux などの中核テクノロジーを使用して Linux コンテナーを実装し、セキュアなマルチテナント機能を実現し、セキュリティーの脆弱性が悪用されるリスクを軽減します。

1.2. LINUX コンテナーのアーキテクチャー

Linux コンテナーを正常に機能させるにはいくつかのコンポーネントが必要になりますが、それらのほとんどは Linux カーネルによって提供されます。カーネルの **名前空間** はプロセスを分離し、**cgroup** を使用するとシステムリソースを制御できます。**SELinux** を使用すると、ホストとコンテナー間の分離や、個々のコンテナー間の分離を確実に実行できます。**管理インターフェース** はこれらのカーネルコンポーネントと対話する上位層を構成し、コンテナーを構築し、管理するためのツールを提供します。

以下の図は、Red Hat Enterprise Linux 7 における Linux コンテナーのアーキテクチャーを示しています。



名前空間

カーネルは、複数のコンテナーに別々の **名前空間** を作成することによりプロセスを分離します。名前空間は、特定のグローバルシステムリソースを抽象化し、名前空間のプロセスに対し、それを分離したインスタンスとして表示させます。そのため、複数のコンテナーが競合せずに同じリソースを同時に使用することができます。名前空間には以下のような種類があります。

- **マウント名前空間** は、プロセスグループが認識するファイルシステムの一連のマウントポイントを分離するため、異なるマウント名前空間にあるプロセスにそれぞれ異なるファイルシステム階層のビューを持たせることができます。マウント名前空間により、mount() および umount() システムコールは、(すべてのプロセスに表示される) マウントポイントのグローバルセットに対する操作を停止し、代わりに該当コンテナープロセスに関連付けられたマウント名前空間のみに影響を与える操作を実行します。たとえば、各コンテナーには独自の **/tmp** または **/var** ディレクトリーか、または完全に異なるユーザー空間を持たせることができます。

- **UTS 名前空間** は、uname() システムコールから返される 2 つのシステム識別子 (nodename および domainname) を分離します。これにより、各コンテナーはそれぞれ独自のホスト名と NIS ドメイン名を持つことができるため、それらの名前に基づいて初期化および設定スクリプトを設定できます。ホストシステムとコンテナーの両方で **hostname** コマンドを実行して異なる結果が出されることを確認することにより、この分離が行われていることをテストすることができます。

- **IPC 名前空間** はシステム V IPC オブジェクトや POSIX メッセージキューなどの特定のプロセス間通信 (IPC) リソースを分離します。つまり、2 つのコンテナーは同じ名前の共有メモリーセグメントやセマフォーを作成できますが、他のコンテナーのメモリーセグメントや共有メモリーと連携することはできません。

- **PID 名前空間** は、異なるコンテナーのプロセスが同じ PID を持てるようにします。これにより、各種のシステム初期化タスクとコンテナーのライフサイクルを管理するために各コンテナーに独自の init (PID1) プロセスを持たせることができます。さらに、各コンテナーにはそれぞれ固有の **/proc** ディレクトリーがあります。コンテナーからは、そのコンテナー内で実行されているプロセスしか監視できないことに注意してください。つまり、コンテナーはそのネイティブなプロセスのみを認識でき、システムの別の部分で実行されているプロセスを「認識」しません。一方、ホストのオペレーティングシステムは、コンテナー内で実行されているプロセスを認識できますが、それらのプロセスに異なる PID 番号を割り当てます。たとえば、ホスト上で **ps -eZ | grep systemd$** コマンドを実行すると、コンテナー内で実行されているものも含む、systemd のすべてのインスタンスを表示できます。

- **ネットワーク名前空間** は、ネットワークコントロラー、ネットワークに関連するシステムリソース、ファイアウォールおよびルーティングテーブルの分離を行います。これにより、コンテナーは別々の仮想ネットワークスタック、ループバックデバイス、およびプロセス空間を使用できます。また、仮想デバイスまたは物理デバイスをコンテナーに追加し、それらに独自の IP アドレスや iptables の完全ルールを割り当てることができます。ホスト上およびコンテナー内の両方で **ip addr** コマンドを実行すると、異なるネットワーク設定を表示できます。

  注意

  ほかにも **ユーザー名前空間** と呼ばれる別のタイプの名前空間があります。ユーザー名前空間は PID 名前空間に似ており、これらを使用して、コンテナーに割り当てられたホストの UID の範囲を指定できます。そのため、プロセスにはコンテナー内の各種操作を実行するための完全 root 特権が割り当てられますが、コンテナー外での操作に対する権限はありません。互換性を維持する理由から、ユーザー名前空間は現行バージョンの Red Hat Enterprise Linux 7 では使用されていませんが、近い将来に有効にされる予定です。

コントロールグループ (cgroup)

カーネルは、システムリソース管理の目的でプロセスをグループ化するために **cgroup** を使用します。cgroup は、CPU 時間、システムメモリー、ネットワーク帯域幅、またはこれらのさまざまな

組み合わせをユーザー定義のタスクグループに割り当てます。Red Hat Enterprise Linux 7 では、cgroup は systemd のスライス、スコープおよびサービスユニットを使用して管理されます。cgroup の詳細は、Red Hat Enterprise Linux 7 リソース管理ガイド を参照してください。

SELinux

SELinux は、SELinux ポリシーとラベルを適用することで、コンテナーのセキュアな分離を行います。また、**sVirt** テクノロジーを使用して、仮想デバイスと統合します。詳細は、Red Hat Enterprise Linux 7 SELinux ユーザーおよび管理者のガイド を参照してください。

管理インターフェース

1.3. SELINUX によるコンテナーの保護

セキュリティー上の理由から、ホストシステムをコンテナーから分離したり、コンテナーを相互に分離したりする必要があります。コンテナーが使用するカーネル機能 (cgoup および名前空間など) はそれ自体である程度のセキュリティーを提供します。cgroup は単一コンテナーがシステムリソースを大量に使用できないようにすることで、一部のサービス拒否攻撃 (DoS) を防ぎます。名前空間により、コンテナー内に作成される **/dev** ディレクトリーは各コンテナーに対して非公開となるため、ホストの変更によって影響を受けることはありません。ただし、システム全体の名前空間が設定されたり、システム全体がコンテナー化される訳ではないため、悪意のあるプロセスがコンテナーから発生することを完全に防ぐことはできません。そのため、SELinux を使用して別の分離レベルを設定する必要があります。

SELinux (Security-Enhanced Linux) は、Linux カーネルにおける強制アクセス制御 (MAC) メカニズム、マルチレベルのセキュリティー (MLS)、およびマルチカテゴリーセキュリティー (MCS) の実装です。**sVirt** プロジェクトは SELinux を基盤として構築されており、Libvirt と統合して仮想マシンとコンテナーの MAC フレームワークを提供します。このアーキテクチャーは、コンテナー内の root プロセスがコンテナー外で実行されている他のプロセスを干渉するのを防ぐため、コンテナーの安全な分離を可能にします。Docker で作成されるコンテナーには、SELinux ポリシーで指定される SELinux コンテキストが自動的に割り当てられます。

デフォルトでは、libvirt ツールで作成されるコンテナーには **virtd_lxc_t** ラベルが割り当てられます (**ps -eZ | grep virtd_lxc_t** を実行)。コンテナー内のプロセスに静的または動的なラベルを設定することにより sVirt を適用することができます。

> **注意**
>
> SELinux がホストシステムで enforcing (強制) モードで実行されている場合であっても、コンテナー内では無効になっている場合があります。これは、ホストとコンテナー内の両方で **getenforce** コマンドを実行して確認することができます。これは、**setenforce** などの SElinux 対応のユーティリティーがコンテナー内で SELinux の動作を実行することを回避するために生じます。

SELinux がホストマシン上で無効にされているか、または permissive (許容) モードで実行されている場合、コンテナーは安全に分離されないことに注意してください。SELinux についての詳細は、Red Hat Enterprise Linux 7 SELinux ユーザーおよび管理者のガイド を参照してください。sVirt については、Red Hat Enterprise Linux 7 仮想化セキュリティーガイド で説明されています。

> **注意**
>
> 現在、Btrfs (B-tree file system) 上に SELinux を有効にした状態でコンテナーを実行することはできません。したがって、推奨される方法として SELinux を有効にした状態で Docker を使用する場合は、**/var/lib/docker** を Brtfs に置かないようにしてください。Brtfs で Docker を実行する必要がある場合

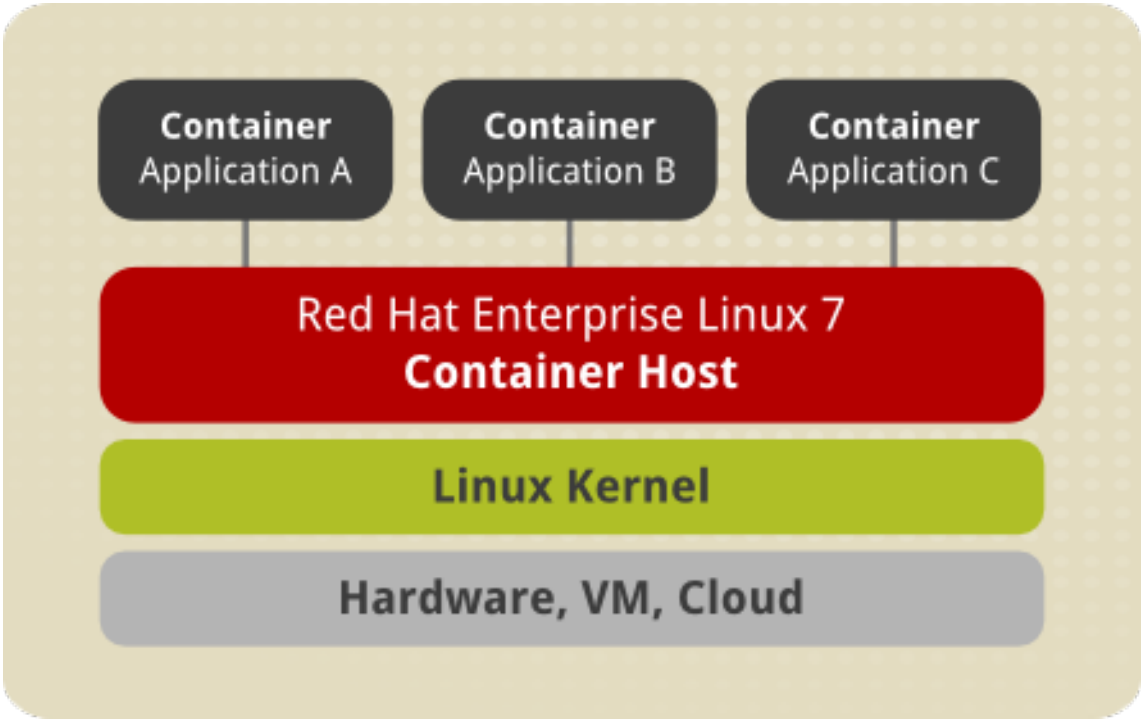
は、**/lib/systemd/system/docker.service** 設定ファイルから **--selinux-enabled** エントリーを削除して SElinux を無効にします。

1.4. コンテナーの使用事例

Red Hat Enterprise Linux 7 で Linux コンテナーを使用する方法として、2 つの一般的なシナリオを紹介します。まずは、**ホストコンテナー** をアプリケーションをサンドボックス化するためのツールとして使用できます。また、**イメージベースのコンテナー** の拡張機能を使用することもできます。

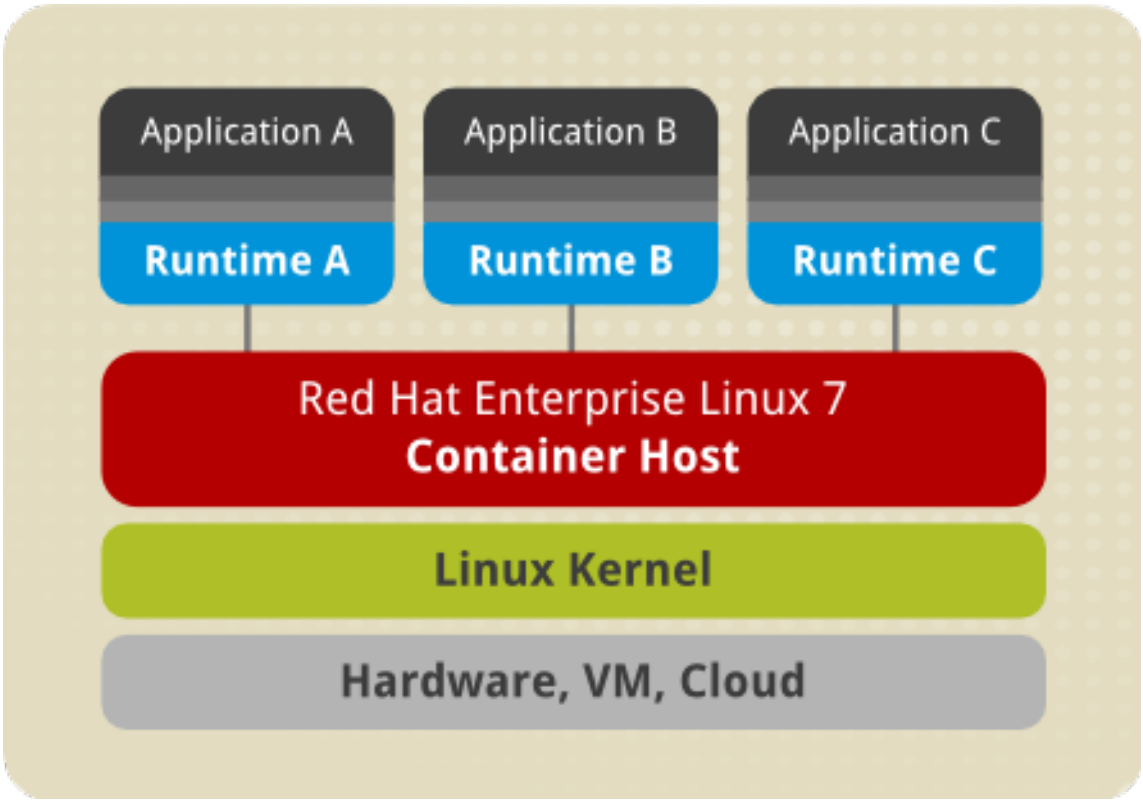
1.4.1. ホストコンテナー

Linux コンテナー機能を搭載した Red Hat Enterprise Linux 7 ホストオペレーティングシステムでは、軽量のアプリケーションサンドボックスとしてコンテナーを作成できます。起動するすべてのホストコンテナーは以下の点で同一です。各コンテナーはホストシステムと同じユーザー空間を実行するため、ホストコンテナー内で実行されるすべてのアプリケーションは Red Hat Enterprise Linux 7 のユーザー空間とランタイムをベースとします。このアプローチの利点は、**yum update** コマンドを使用して、セキュリティーエラータや他のアップデートをこれらのコンテナーに簡単に適用できる点にあります。



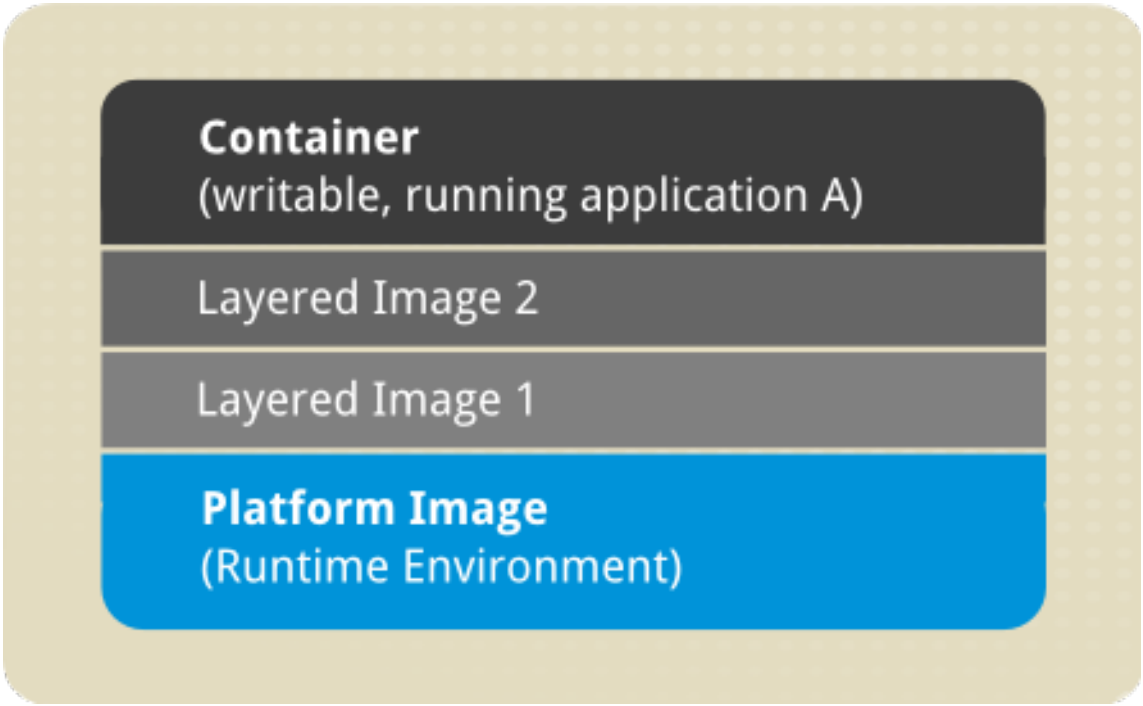
1.4.2. イメージベースのコンテナー

イメージベースのコンテナーでは、アプリケーションは個別のランタイムスタックでパッケージングされます。これにより、アプリケーションをホストオペレーティングシステムから切り離すことができます。また、異なるプラットフォーム用に開発されたアプリケーションの複数のインスタンスを実行することができます。これはコンテナーのランタイムとアプリケーションのランタイムがイメージ形式でデプロイされるために可能になります。たとえば、図にある Runtime A が Red Hat Enterprise Linux 6.5 であることも、Runtime B がバージョン 6.6 である可能性もあります。



イメージベースのコンテナーは、アプリケーションの複数のインスタンスとバージョンをホストし、オーバーヘッドを最小化し、柔軟性を高めることができます。このようなコンテナーはホスト固有の設定に関連付けられないため、移植できます。

Docker フォーマットは、Red Hat Enterprise Linux 7 にコピーオンライト (COW) を実装するための LVM スナップショットの高度な機能である **device mapper のシンプロビジョニング** 技術を使用しています。



上の図は、イメージベースのコンテナーの基本的なコンポーネントを示しています。

- **コンテナー** (狭義) – アプリケーションが実行されるアクティブなコンポーネントです。各コンテナーは、必要な設定データを保持する **イメージ** をベースとしています。イメージからコンポーネントを起動すると、書き込み可能な層がこのイメージの上部に追加されます。コンテ

ナーのコミット (**`docker commit`** コマンドを使用) を実行するたびに、変更を保存するためのイメージ層が新たに追加されます。

- **イメージ** - コンテナー設定の静的なスナップショットです。イメージは変更されない読み取り専用層になります。すべての変更は最上位の書き込み可能な層で行われ、保存は新しいイメージを作成することで行われます。それぞれのイメージは、1 つまたは複数の親イメージによって異なります。
- **プラットフォームイメージ** – 親を持たないイメージです。プラットフォームイメージは、コンテナー化されたアプリケーションの実行に必要なランタイム環境、パッケージおよびユーティリティーを定義します。通常、Docker の使用はプラットフォームイメージを取得するところから始まります。プラットフォームイメージは読み取り専用であるため、すべての変更はプラットフォームイメージの上部に重ねられるコピーイメージに反映されます。Red Hat は、Red Hat Enterprise 6 のほかにも Red Hat Enterprise Linux 7 のプラットフォームイメージを提供しています。

プラットフォームイメージの上部に重ねられるイメージは、コンテナー化されたアプリケーションのソフトウェア依存関係を含む **アプリケーション層** を作成します。たとえば、重ねられたイメージは、コンテナー化されたアプリケーションが必要とするソフトウェアの依存関係を追加している可能性があります。

アプリケーション層に含まれるパッケージの数によって、コンテナーの全体サイズは非常に大きくなるか、または非常に小さくなる可能性があります。さらに、サードパーティーの独立ソフトウェアベンダーのソフトウェアなどの、イメージの層をさらに重ねることができます。ユーザーからは 1 つのコンテナーがあるように見えますが、運用上は、その背後に多くの層が存在する場合があります。

1.5. LINUX コンテナーと KVM 仮想化の比較

KVM 仮想マシンは、独自のカーネルを必要とします。Linux コンテナーはホストオペレーティングシステムのカーネルを共有します。通常、同じハードウェアで、仮想マシンよりもはるかに多くの数のコンテナーを起動することができます。

Linux コンテナーと KVM 仮想化にはそれぞれの長所と短所があるため、どちらの技術がより一般的に選択されるかはその使用事例によって異なります。

KVM 仮想化:

- KVM 仮想化は、Linux 以外のシステムも含め、様々な種類のフルオペレーティングシステムの起動を可能にします。仮想マシンのリソース集約型の性質 (コンテナーとの対比) は、ホストで実行できる仮想マシンの数が同じホストで実行できるコンテナーの数よりも少なくなることを示します。
- 通常カーネルインスタンスを別々に実行することにより、分離が行われ、セキュリティー機能が提供されます。いずれかのカーネルが予期せずに終了しても、システム全体が無効になることはありません。
- ゲスト仮想マシンはホストの変更の影響を受けません。そのため、ホストと仮想マシンで同じアプリケーションの異なるバージョンをそれぞれ実行することができます。さらに KVM はライブマイグレーションなどの数多くの便利な機能を提供します。これらの機能の詳細は、Red Hat Enterprise Linux 7 仮想化の導入および管理ガイド を参照してください。

Linux コンテナー:

- Linux コンテナーは 1 つ以上のアプリケーションの分離をサポートするために設計されています。

- システム全体の変更は各コンテナーで確認できます。たとえば、ホストマシンでアプリケーションをアップグレードすると、この変更はこのアプリケーションの各種インスタンスを実行するすべてのサンドボックスにも適用されます。
- コンテナーは軽量であるため、多数のコンテナーをホストマシンで同時に実行することができます。理論上、コンテナーの最大数は 6000で、root のファイルシステムディレクトリーのバインドマウント数は 12,000 です。

1.6. その他のリソース

Linux コンテナーの一般的な原則およびアーキテクチャーについての詳細は、以下のリソースを参照してください。

インストールされているドキュメント

- docker(1): **`docker`** コマンドの man ページ。

オンラインドキュメント

- Red Hat Enterprise Linux 7 仮想化の導入および管理ガイド: このガイドは、Red Hat Enterprise Linux 7 ホストの物理マシンを設定する方法、および KVM ハイパーバイザーを使用して、各種のディストリビューションにゲストの仮想マシンをインストールし、設定する方法について説明しています。さらに、PCI デバイス設定、SR-IOV、ネットワーク設定、ストレージ、デバイスおよびゲスト仮想マシンの管理や、トラブルシューティング、互換性および制限について説明しています。
- Red Hat Enterprise Linux 7 SELinux ユーザーおよび管理者のガイド: Red Hat Enterprise Linux 7 の『SELinux ユーザーおよび管理者のガイド』では、SELinux の機能に必要な基本要素と原則、および各種サービスをセットアップし、設定するための実際的なタスクについて説明しています。
- Docker フォーマットのコンテナーイメージの使用方法: このクィックスタートガイドは、Docker 関連の必須タスクを数多くの例を使って説明しています。
- Docker プロジェクトのドキュメントサイト: Docker プロジェクトの公式ドキュメントです。

第2章 KUBERNETES によるコンテナーのオーケストレーション

2.1. 概要

Kubernetes は、Docker コンテナーのオーケストレーションおよび管理を行うためのツールです。以下の手順では、単一システムの Kubernetes サンドボックスをセットアップします。

- Kubernetes で、2 つのコンテナーおよび yaml ファイルを使用し、コンテナーをデプロイする。
- Kubernetes でコンテナーを管理する。

この手順では、Kubernetes を実際に使用し、その機能を学習するためのサンドボックスをセットアップします。この手順では、通常は別々の Kubernetes Master システムや複数の Kubernetes Node システムにあるサービスが単一システムで実行されます。

2.2. KUBERNETES について

Docker はコンテナーフォーマットを定義し、個々のコンテナーを構築し、管理しますが、オーケストレーションツールは一連のコンテナーをデプロイし、管理するために必要です。Kubernetes は Docker コンテナーのオーケストレーションを行うためのツールです。必要なコンテナーイメージを構築した後に、Kubernetes Master を使用して Pod に 1 つ以上のコンテナーをデプロイすることができます。Master はその Pod のコンテナーを、コンテナーが実行される Kubernetes Node にプッシュします。

たとえば、Kubernetes Master および Node はどちらも同じコンピューターにあり、そのコンピューターは RHEL 7 Server または RHEL 7 Atomic Host のいずれかであるとします。Kubernetes は Kubenetes Master および Node の機能を実装するために一連サービスデーモンに依存します。Kubernetes Master および Node については以下を理解しておく必要があります。

- **Master**: Kubernetes Master では、API 呼び出しを Kubernets クラスターの Pod、レプリケーションコントローラー、サービス、ノードその他のコンポーネントを制御するサービスに指定します。通常これらの呼び出しは、**kubectl** コマンドを実行して行われます。コンテナーは Master から Node 上で実行されるようにデプロイされます。
- **Node**: Node はコンテナーのランタイム環境を提供するシステムです。

Pod の定義は設定ファイル (yaml または json 形式) に保存されます。以下の手順を使用して、単一の RHEL 7 または RHEL Atomic システムをセットアップし、これを Kubernetes Master および Node として設定し、yaml ファイルを使用して Pod に各コンテナーを定義し、それらのコンテナーを Kubernetes を使用してデプロイします (**kubectl** コマンド)。

2.3. KUBERNETES POD からのコンテナーの実行

Docker コンテナーを構築し、Kubernetes を使用してオーケストレーションを行うには RHEL 7 または RHEL Atomic システムが必要です。Kubernetes Master および Node システムには、異なるサービスデーモンのセットが必要です。以下の手順では、両方のサービスデーモンのセットが同じシステムで実行されます。

コンテナー、システムおよびサービスを配置したら、**kubectl** コマンドを使用し、それらのコンテナーが Kubernetes Node (この場合はローカルシステム) で実行されるようにデプロイします。

以下にその手順を示します。

2.3.1. Kubernetes で Docker コンテナーをデプロイするためのセットアップ

Kubernetes を準備するには、RHEL 7 または RHEL Atomic をインストールし、firewalld を無効にしてから 2 つのコンテナーを取得する必要があります。

1. **RHEL 7 または RHEL Atomic システムのインストール**: この Kubernetes サンドボックスシステムについては、まず RHEL 7 または RHEL Atomic システムをインストールし、システムのサブスクライブを実行してから、Docker サービスのインストールおよび起動を行います。基本的な RHEL または RHEL Atomic システムをセットアップして Kubernetes で使用する方法の詳細は、以下を参照してください。

   Get Started with Docker Formatted Container Images on Red Hat Systems (Docker フォーマットのコンテナーイメージの使用方法)

2. **Kubernetes のインストール**: RHEL 7 システムの場合は、kubernetes および etcd パッケージをインストールします。これにより、kubernetes-node および kubernetes-master パッケージも取得できます (これらのパッケージは RHEL Atomic にすでにインストールされています)。

   ```
   # yum install kubernetes etcd
   ```

3. **firewalld の無効化**: RHEL 7 ホストを使用している場合は firewalld サービスが無効にされていることを確認します (firewalld サービスは Atomic Host にはインストールされていません)。RHEL 7 では以下を入力して firewalld サービスを無効にし、これを停止します。

   ```
   # systemctl disable firewalld
   # systemctl stop firewalld
   ```

4. **Docker コンテナーの取得**: RHEL Atomic Host のスタートガイドの指示に従って以下の 2 つのコンテナーを構築し、それらのコンテナーのイメージを両方のノードで利用できるようにします (つまり、いずれかのノードで docker images コマンドを入力する際に 2 つのイメージが表示されるようにします)。

   - Simple Apache Web Server in a Docker Container (簡易 Apache Web サーバーをコンテナーに構築する)
   - Simple Database Server in a Docker Container (簡易データベースサーバーをコンテナーに構築する)

   コンテナーの構築およびテストを実行した後は、停止します (**docker stop mydbforweb** および **docker stop mywebwithdb** を実行)。

2.3.2. Kubernetes の起動

Kubernetes Master および Node サービスはどちらもローカルシステム上で実行されるため、Kubernetes 設定ファイルを変更する必要はありません。Master および Node サービスはローカルホストで相互を参照し、サービスはローカルホストでのみ利用可能になります。

1. **apiserver ファイルの設定**: デフォルトで kube-apiserver サービスの許可制御機能を使用するには、Pod を起動するすべてのユーザー用にアカウントをセットアップする必要があります。これにより、Kubernetes プロバイダーは、ユーザーアカウント別に使用を追跡し、制限することができます。この一体型の例では、**/etc/kubernetes/apiserver** ファイルを開き、KUBE_ADMISSION_CONTROL 値を変更して --admission_control オプションから

"ServiceAccount" を削除することでこの機能を無効にすることができます。以下の行は、元の行 (コメントアウトされた行) および ServiceAccount が削除された新規の行を示しています (これらの 2 つの行は折り返されますが、2 行に分割されているように表示される場合もあります)。

```
# KUBE_ADMISSION_CONTROL="--
admission_control=NamespaceLifecycle,NamespaceExists,LimitRanger,
SecurityContextDeny,ServiceAccount,ResourceQuota"
KUBE_ADMISSION_CONTROL="--
admission_control=NamespaceLifecycle,NamespaceExists,LimitRanger,
SecurityContextDeny,ResourceQuota"
```

2. **Kubernetes Master サービスデーモンの起動**: Kubernetes Master に関連付けられたいくつかのサービスを起動する必要があります。

```
# for SERVICES in etcd kube-apiserver kube-controller-manager
kube-scheduler; do
    systemctl restart $SERVICES
    systemctl enable $SERVICES
    systemctl status $SERVICES
done
```

3. **Kubernetes Node サービスデーモンの起動**: Kubernetes Node に関連付けられたいくつかのサービスを起動する必要があります。

```
# for SERVICES in docker kube-proxy.service kubelet.service; do
    systemctl restart $SERVICES
    systemctl enable $SERVICES
    systemctl status $SERVICES
done
```

4. **サービスの確認**: ss コマンドを実行して、サービスが実行されているポートを確認します。

```
# ss -tulnp | grep -E "(kube)|(etcd)"
```

5. **etcd サービスのテスト**: 以下のように curl コマンドを使用して etcd サービスを確認します。

```
# curl -s -L http://localhost:2379/version
{"etcdserver":"2.1.1","etcdcluster":"2.1.0"}
```

注意: RHEL Atomic Host 7.1.4 以降では、etcd ユーティリティーはポート 2379 を使用します。古いバージョンの RHEL Atomic Host を実行している場合は、上記のコマンドの 2379 を 4001 に置き換えます。

2.3.3. Kubernetes によるコンテナー Pod の起動

ローカルシステムで Master および Node サービスを実行し、2 つのコンテナーイメージを配置したら、Kubernetes Pod を使用してコンテナーを起動できます。以下はいくつかの注意点です。

- **複数の Pod**: 単一の Pod で複数のコンテナーを起動できますが、それらのコンテナーを別々の Pod に配置すると、各コンテナーは他のコンテナーを起動することなく、複数のインスタンスを必要に応じて複製できます。

- **Kubernetes サービス**: この手順では、データベースおよび Web サーバーの Pod 用に Kubernetes サービスを定義し、コンテナーが Kubernetes からそれらのサービスを見つけられるようにします。データベースおよび Web サーバーは、ターゲット Pod が実行されている IP アドレス、ポート番号、またはノードを認識しない場合でも相互を識別できます。

以下の手順は、2 つの Pod を起動し、テストする方法について説明しています。

重要: yaml ファイルのインデントを維持することは重要です。yaml ファイルのスペースは、フォーマットを整理された状態に保つために使用されます (構造を維持するための波括弧などの文字は不要です)。

1. **データベース Kubernetes サービスの作成**: **db-service.yaml** ファイルを作成して、データベースサービスを Kubernetes に提供する Pod を特定します。

```
Kubernetes:
apiVersion: v1
kind: Service
metadata:
  labels:
    name: db
  name: db-service
  namespace: default
spec:
  ports:
  - port: 3306
  selector:
    name: db
```

2. **データベースサーバーのレプリケーションコントローラーファイルの作成**: データベースサーバーの Pod をデプロイするために使用する **db-rc.yaml** ファイルを作成します。以下を設定します。

```
kind: "ReplicationController"
apiVersion: "v1"
metadata:
  name: "db-controller"
spec:
  replicas: 1
  selector:
    name: "db"
  template:
    spec:
      containers:
        - name: "db"
          image: "dbforweb"
          ports:
            - containerPort: 3306
    metadata:
      labels:
        name: "db"
      selectorname: "db"
  labels:
    name: "db"
```

3. **Web サーバーの Kubernetes サービスファイルの作成**: Web サーバーの Pod をデプロイするために使用する **webserver-service.yaml** ファイルを作成します。以下を設定しま

す。

```
Kubernetes:
apiVersion: v1
kind: Service
metadata:
  labels:
    name: webserver
  name: webserver-service
  namespace: default
spec:
  ports:
  - port: 80
  selector:
    name: webserver
```

4. **Web サーバーのレプリケーションコントローラーファイルの作成**: Web サーバーの Pod をデプロイするために使用する **webserver-rc.yaml** ファイルを作成します。以下を設定します。

```
kind: "ReplicationController"
apiVersion: "v1"
metadata:
  name: "webserver-controller"
spec:
  replicas: 1
  selector:
    name: "webserver"
  template:
    spec:
      containers:
        - name: "apache-frontend"
          image: "webwithdb"
          ports:
            - containerPort: 80
    metadata:
      labels:
        name: "webserver"
        uses: db
  labels:
    name: "webserver"
```

5. **kubectl を使用したコンテナーのオーケストレーション**: 現在のディレクトリーにある 2 つの yaml ファイルを使用して以下のコマンドを実行し、コンテナーの実行を開始するために Pod を起動します。

```
# kubectl create -f db-service.yaml
services/db-service
# kubectl create -f db-rc.yaml
replicationcontrollers/db-controller
# kubectl create -f webserver-service.yaml
services/webserver-service
# kubectl create -f webserver-rc.yaml
replicationcontrollers/webserver-controller
```

6. **rc、pod、およびサービスの確認**: 以下のコマンドを実行し、レプリケーションコントローラー、Pod、およびサービスがすべて実行されていることを確認します。

```
# kubectl get rc
CONTROLLER             CONTAINER(S)      IMAGE(S)    SELECTOR
REPLICAS
db-controller          db                dbforweb    name=db
1
webserver-controller   apache-frontend   webwithdb
name=webserver   1
# kubectl get pod
NAME                          READY     STATUS    RESTARTS   AGE
db-controller-tiu56           1/1       Running   0          48m
webserver-controller-s5b81    1/1       Running   0          33m
# kubectl get service
NAME                LABELS          SELECTOR          IP(S)
PORT(S)
db-service          name=db         name=db
10.254.232.194   3306/TCP
kubernetes          component=...   <none>            10.254.0.1
443/TCP
webserver-service   name=webserver  name=webserver
10.254.104.45    80/TCP
```

7. **コンテナーの確認**: 2 つのコンテナーがどちらも実行中で、Web サーバーコンテナーがデータベースサーバーを認識する場合、以下のように kubectl コマンドを使って Pod を確認し、curl コマンドを実行してすべてが機能していることを確認できます。

```
# kubectl get pods
# curl http://localhost:80/cgi-bin/action
<html>
<head>
<title>My Application</title>
</head>
<body>
<h2>RedHat rocks</h2>
<h2>Success</h2>
</body>
</html>
```

ローカルホストに Web ブラウザーをインストールしている場合は、Web ブラウザーを開いて、出力の行がきちんと表示されていることを確認します。ブラウザーで URL の **http://localhost/cgi-bin/action** を開きます。

2.4. KUBERNETES POD の確認

問題が発生した場合には、原因を判別するためのいくつかの方法があります。たとえば、コンテナー内のサービスを調べることができます。これを実行するには、コンテナー内のログを調べて原因を確認できます。以下のコマンドを実行します (最後の引数を、確認する Pod の名前に置き換えます)。

そのほかにも、firewalld を無効にしなかったことが問題の原因となることがあります。firewalld がアクティブな場合は、サービスがコンテナー間でのポートへのアクセスを試行する際にポートへのアクセスがブロックされる可能性があります。ホスト上で必ず **systemctl stop firewalld ; systemctl disable firewalld** を実行してください。

2 つの Pod を持つアプリケーションの作成時に間違いがあった場合、レプリケーションコントローラーとサービスを削除することができます (Pod はレプリケーションコントローラーが削除されるとなくなります)。その後に、yaml ファイルを修正し、それらを作成し直すことができます。以下に、レプリケーションコントローラーおよびサービスを削除する方法を示します。

```
# kubectl delete rc webserver-controller
replicationcontrollers/webserver-controller
# kubectl delete rc db-controller
replicationcontrollers/db-controller
# kubectl delete service webserver-service
services/webserver-service
# kubectl delete service db-service
```

Pod だけ削除することがないようにしてください。レプリケーションコントローラーを削除せずに Pod を削除すると、レプリケーションコントローラーは新規の Pod を起動して削除した Pod の置き換えを行います。

上記の例は、簡単に Kubernetes を起動する方法です。ただし、同じシステムで Master と Node を 1 つずつ使用しているだけなので、拡張性がありません。拡張性のある設定をするには、Kubernetes クラスターを作成して Docker フォーマットのコンテナーイメージを実行する に記載の手順に従って設定することが推奨されます。

第3章 KUBERNETES クラスターを作成して DOCKER フォーマットのコンテナーイメージを実行する

3.1. 概要

Kubernetes を使用すると、1 つのシステム (Kubernetes Master) の Docker フォーマットのコンテナーを実行し、管理することができ、それらのコンテナーを他のシステム (Kubernetes Node) にデプロイし、実行することができます。Kubernetes 自体はシステムサービスのセットで構成されており、これにより、**Pod** と呼ばれる場所で Docker コンテナーを起動し、管理することができます。それらの Pod は複数のノードで実行でき、Kubernetes **サービス** によって相互に接続できます。Pod はいったん定義されると、ニーズに合わせて簡単に拡張/縮小することができます。

以下の手順では、次の作業を行います。

- 1 つの Kubernetes Masterと 2 つの Kubernetes Node (minion とも呼ばれる) として使用する 3 つの RHEL Atomic ホストまたは Red Hat Enterprise Linux 7 システムをセットアップ。
- Flannel を使用したコンテナー間のネットワークの設定。
- サービス、Pod およびレプリケーションコントローラーを定義するデータファイル (yaml 形式) の作成。

この手順で起動するサービスには、etcd、kube-apiserver、kubecontroller-manager、kube-proxy、kube-scheduler、および kubelet が含まれます。yaml ファイルからコンテナーを起動したり、その他の方法で Kubernetes 環境を管理したりするには、**kubectl** コマンドを実行します。以下の手順では、以下のいずれかのシステムが実行されていることを仮定しています。

- **RHEL 7.2 Atomic Host**: RHEL 7.2 を使用していない場合は、**atomic host upgrade** を実行して RHEL Atomic host を再起動します。
- **RHEL 7.2 サーバー**: RHEL サーバーには少なくとも以下のバージョンが含まれている必要があります (含まれていない場合は **yum upgrade kubernetes etcd docker flannel** を実行します)。
  - kubernetes-1.2
  - etcd-2.2
  - docker-1.6
  - flannel-0.2

本書の改訂前のバージョンを使用している場合は、設定をこの最新ソフトウェアにアップグレードする方法について「Upgrading a Kubernetes Cluster」という記事を参照してください。

注意: Kubernetes および関連サービスは RHEL Atomic Host に組み込まれています。Atomic の長期的な目標は可能な限りスリム化することであるため、Kubernetes ソフトウェアは最終的にはベースの RHEL Atomic システムから削除され、Docker フォーマットコンテナーのセットに組み込まれる予定です。この移行に役立つ機能として、Kubernetes Master をセットアップするためのほとんど機能はコンテナーで利用できます。そのため、この手順の実行過程でコンテナーから一部の Kubernetes Master サービスを実行することを選択できます。

3.2. KUBERNETES によるコンテナーのデプロイ準備

まずは、最新の RHEL Atomic Host または RHEL 7 Server インストールメディアを取得して、3 つのノードのそれぞれにそれらのオペレーティングシステムの 1 つをインストールする必要がありま

す。「Red Hat Enterprise Linux Atomic Host スタートガイド」にある指示では、ベアメタルか、またはいくつかの異なる仮想環境のいずれかで実行するために Atomic をセットアップする方法についての情報を提供しています。その後、Kubernets をセットアップする前にいくつかのシステム設定を行う必要があります。

1. **インストールメディアの取得**: 以下のように RHEL Atomic Host または RHEL Server メディアを取得します。
   - RHEL Atomic Host 7.2 メディア: Red Hat Enterprise Linux Downloads に移動し、利用可能なメディアから選択して、RHEL Atomic Host 7.2 メディアを取得します。
   - RHEL 7.2 Server メディア: Red Hat Enterprise Linux Downloads に移動し、利用可能なメディアから選択して、Red Hat Enterprise Linux 7.2 Server メディアを取得します。
2. **RHEL または RHEL Atomic Host システムのインストール**: この手順では、3 つ以上の RHEL Atomic Host システムがインストールされる必要があります。インストール時は以下を実行してください。
   - ネットワークインターフェースを設定し、インターフェースが Kubernetes クラスターの他のシステムと通信できるようにします。
   - 使用するコンテナーのサイズと数に対応する十分なディスク領域を確保します。
   - インストールの完了時にシステムを再起動し、追加の設定を実行します。
3. **NTP (Network Time Protocol) の設定**: クラスター (Master および Node) のすべてのシステムで、時刻が同期されている必要があります。それらのシステムに Network Time Protocol デーモン (ntpd) または Chrony デーモン (chronyd) をセットアップする方法については、『RHEL 7 システム管理者のガイド』を参照してください。
4. **DNS または /etc/hosts の設定**: システムが DNS で相互に接続できるようにするか、またはそれらの名前および IP アドレスを **/etc/hosts** に追加します。たとえば、各システムの **/etc/hosts** ファイルにあるエントリーは以下のように表示されます。

```
192.168.122.119   master
192.168.122.196   node1
192.168.122.27    node2
```

5. **各システムのサブスクライブ**: 以下のように **subscription-manager** コマンドを使用して 3 つのシステムすべてのサブスクライブを行います。

```
# subscription-manager register --auto-attach --username=
<rhnuser> --password=<password>
```

6. **各システムのアップグレード**: 使用しているのが RHEL Server または RHEL Atomic Host システムであるかに応じて、最新ソフトウェアへのアップグレードの方法は異なります。さらに、RHEL Atomic Host では Docker がすでにインストールされていますが、RHEL Server で Docker を取得するには、2 つのリポジトリーを有効にし、Docker パッケージをインストールする必要があります。

   RHEL Atomic Host システムで、以下を入力します。

```
# atomic host upgrade
# reboot
```

   RHEL Server システムで、以下を入力します。

```
# yum upgrade -y
# subscription-manager repos --enable=rhel-7-server-extras-rpms
# subscription-manager repos --enable=rhel-7-server-optional-rpms
# yum install docker
```

7. **firewalld の無効化**: RHEL 7 ホストを使用している場合、firewalld サービスが無効にされていることを確認します (firewalld サービスは RHEL Atomic Host にはインストールされていません)。RHEL 7 で以下を入力し、firewall サービスを無効にし、これを停止します。

```
# systemctl disable firewalld
# systemctl stop firewalld
```

8. **Docker の起動および有効化**: Docker サービスが各システムで実行されていることを確認するには (マスターのコンテナーから Kubernetes を実行する場合はマスターを含む)、以下を実行します。

```
# systemctl restart docker
# systemctl enable docker
# systemctl status docker
* docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service;
enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/docker.service.d
           |-flannel.conf
   Active: active (running) since Wed 2015-11-04 15:37:10 EST;
22h ago
...
```

9. **Docker コンテナーの取得**: 以下にある説明に従って 2 つのイメージを構築します。「簡易 Apache Web サーバーをコンテナーに構築する」および「簡易データベースサーバーをコンテナーに構築する」

10. **コンテナーの各ノードへのインポート**: 各イメージを各ノードにコピーし、それらを各ノードにロードします。イメージを構築したシステムから開始して、以下を実行します。

```
# docker export mydbforweb > dbforweb.tar
# docker export mywebwithdb > webwithdb.tar
# scp dbforweb.tar webwithdb.tar node1:/tmp
# scp dbforweb.tar webwithdb.tar node2:/tmp
```

各ノードで以下を実行します。

```
# cat /tmp/webwithdb.tar | docker import - webwithdb
# cat /tmp/dbforweb.tar | docker import - dbforweb
# docker images | grep web
dbforweb  latest  2e48be49fec4  About a minute ago  568.7 MB
webwithdb latest  5f2daa2e04e9  2 minutes ago       408.6 MB
```

注意: コンテナーを各ノードにインポートする代わりに、プライベートのコンテナーレジストリーを作成し、後で作成する yaml ファイルが、コンテナーの起動時にレジストリーを参照するようにできます。

コンテナーの構築およびテストを実行した後は、停止します (**docker stop mydbforweb** および **docker stop mywebwithdb** を実行)。

3 つの RHEL Atomic Host の基本的なセットアップが完了しました。Kubernetes のセットアップを開始することができます。

3.3. KUBERNETES のセットアップ

これらの 3 つのシステムが配置されたら、次に Kubernetes をセットアップします。この手順は、以下の条件によって若干異なります。

- マスターまたはノードのセットアップ
- RHEL Server または RHEL Atomic Host システムの使用
- ホスト上の systemd サービスから、またはホスト上のコンテナーからのマスターサービスの実行 (ノードサービスのコンテナー化されたバージョンはまだ利用できません)

重要: ノードを起動する前にマスターを設定し、起動するようにしてください。マスターを起動する前にノードを起動すると、ノードを Kubernetes に適切に登録できない場合があります。

3.4. MASTER での KUBERNETES のセットアップ

以下の手順は、Kubernetes Master 上でサービスをセットアップし、実行する方法について説明しています。

1. **Kubernetes のインストール**: RHEL 7 システムをお使いの場合、kubernetes パッケージおよび etcd パッケージをマスターにインストールしてください (これらは RHEL Atomic にすでにインストールされています)。

```
# yum install kubernetes-master etcd
```

2. **etcd サービスの設定**: **/etc/etcd/etcd.conf** を編集します。etcd サービスは、すべてのインターフェースでポート 2380 (ETCD_LISTEN_PEER_URLS) およびポート 2379 (ETCD_LISTEN_CLIENT_URLS) を listen し、ローカルホストの 2380 (ETCD_LISTEN_PEER_URLS) を listen するように設定される必要があります。このファイルのコメントが外されている行は以下のように表示されます。

```
ETCD_NAME=default
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:2379"
ETCD_LISTEN_PEER_URLS="http://localhost:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://0.0.0.0:2379"
```

3. **Kubernete 設定ファイルの編集**: **/etc/kubernetes/config** ファイルを編集し、KUBE_MASTER 行を変更してマスターサーバーの場所を特定します (デフォルトでは 127.0.0.1 を参照します)。他の設定はそのままにします。変更される行は以下のように表示されます。

```
KUBE_MASTER="--master=http://master.example.com:8080"
```

4. **Kubernetes apiserver の設定**: **/etc/kubernetes/apiserver** を編集し、新規の KUBE_ETCD_SERVERS 行 (下記) を追加し、apiserver 設定ファイルの他の行を確認し、変更します。KUBE_API_ADDRESS を変更して、ローカルホストだけではなく、すべてのネットワークアドレス(0.0.0.0) で listen するようにします。Kuberntes がサービスに割り

当てるように使用できる KUBE_SERVICE_ADDRESS のアドレス範囲を設定します (後述するこのアドレスの説明を参照してください)。最後に "ServiceAccount" という項目を KUBE_ADMISSION_CONTROL 命令から削除します (以下は、新規の行です。その上の元の行はコメントアウトされています)。以下は例になります。

```
KUBE_API_ADDRESS="--insecure-bind-address=0.0.0.0"
KUBE_ETCD_SERVERS="--etcd_servers=http://127.0.0.1:2379"
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16"
# KUBE_ADMISSION_CONTROL="--
admission_control=NamespaceLifecycle,NamespaceExists,LimitRanger,
SecurityContextDeny,ServiceAccount,ResourceQuota"
KUBE_ADMISSION_CONTROL="--
admission_control=NamespaceLifecycle,NamespaceExists,LimitRanger,
SecurityContextDeny,ResourceQuota"
```

以下は、使用する KUBE_SERVICE_ADDRESSES アドレス範囲のいくつかの注意点です。

- アドレス範囲は、Kubernetes が Kubernetes サービスに割り当てるために使用します。
- この例では、アドレス範囲 10.254.0.0/16 は、必要に応じて Kubernets が割り当てることのできる 10.254 サブネットのセットを使用します。たとえば、10.254.1.X、10.254.2.X などになります。
- このアドレス範囲が環境内の他の場所では使用されません。
- 割り当てられる各アドレス範囲はノード内でのみ使用され、そのノード外ではルーティングすることはできません。
- このアドレス範囲は flannel が使用する範囲とは異なります (flannel のアドレス範囲は Pod に割り当てられます)。

5. **マスターサービスの起動**: kubernetes サービスは、ホストのファイルシステムまたはコンテナーのいずれかから実行することを選択できます。**etcd** サービスは現時点ではホストから実行する必要があります。どの方法にするかを定めてから、以下の該当するコマンドに従ってください。

   - **ホストから実行**: Kubernetes Master サービスをホストのファイルシステムから直接実行するには、複数の systemd サービスを有効にし、起動する必要があります。マスターから以下の **for** ループを実行して、マスターで Kubernetes systemd サービスを起動し、有効にします。

     ```
     # for SERVICES in etcd kube-apiserver kube-controller-manager
     kube-scheduler; do
         systemctl restart $SERVICES
         systemctl enable $SERVICES
         systemctl status $SERVICES
         done
     ```

   - **コンテナーから実行**: (systemctl を使用して) ホストから Kubernetes Master サービスを実行する代わりに、コンテナーからそれらを実行することができます。Kubernetes を完全にコンテナー化することが長期的な目標になりますが、一部のコンポーネントは現在も開発中です。この手順は、コンテナーから実行される Kubernetes サービスを手動で試行する方法を示しています。「Running Kubernetes from Containers」の記事を参照し、手順が実行可能になり次第利用できるより永続的なソリューションを確認してください。

     コンテナー化された Kubernetes Master サービスを実行するには、サービスが停止して

おり、systemd から無効にされていることを確認する必要があります。次に、以下のようにコンテナーサービスを設定する必要があります。

- **Kubernetes Master の systemd サービスの無効化**: マスターで以下の **for** ループを実行します。結果として、すべてのシンボリックリックが削除され、すべてのサービスで非アクティブまたは失敗と表示されるはずです。**etcd** を systemd サービスとして使用します。

```
# for SERVICES in kube-apiserver kube-controller-manager
kube-scheduler; do
    systemctl stop $SERVICES
    systemctl disable $SERVICES
    systemctl is-active $SERVICES
    done
# systemctl restart etcd ; systemctl enable etcd
```

- **Kubernetes Master コンテナーの取得**: 必要なコンテナーを取得するには、以下を実行します。

```
# docker pull rhel7/kubernetes-controller-mgr
# docker pull rhel7/kubernetes-apiserver
# docker pull rhel7/kubernetes-scheduler
```

- **Kubernetes Master サービスの起動**: マスターで kubernetes サービスを起動するには、以下を入力します。

```
# docker run -v /etc/kubernetes/:/etc/kubernetes --net=host -d
\
   rhel7/kubernetes-apiserver

# docker run -v /etc/kubernetes/:/etc/kubernetes --net=host -d
\
   rhel7/kubernetes-controller-mgr

# docker run -v /etc/kubernetes/:/etc/kubernetes --net=host -d
\
   rhel7/kubernetes-scheduler
```

- **Kubernetes Master サービスの確認**: kubernetes サービスが systemd サービスとして実行されていないため、**systemctl** を使用してサービスが実行中であることを確認することはできません。それらが実行されていることを確認するには、**ps** コマンドを実行できます。

```
# ps -ef | grep kube
root  2566   1507  0 11:26 ?  00:00:26 /usr/bin/kube-apiserver --
logtostderr=true ...
root  2619   1507  0 11:26 ?  00:00:13 /usr/bin/kube-controller-
manager ...
root  2671   1507  0 11:26 ?  00:00:01 /usr/bin/kube-scheduler --
logtostderr=true ...
```

3.5. NODE での KUBERNETES のセットアップ

2 つの Kubernetes ノード (この例では node1.example.com および node2.example.com) のいずれかで、いくつかの設定ファイルを編集し、複数の Kubernetes systemd サービスを起動し、有効にする必要があります。

1. **Kubernetes のインストール**: ノードが RHEL 7 システムの場合、各ノードに kubernetes-node パッケージをインストールします。Kubernetes は RHEL Atomic にすでにインストールされています。

```
# yum install kubernetes-node
```

2. **/etc/kubernetes/config の編集**: このファイルの KUBE_MASTER 行を編集し、マスターの位置 (デフォルトでは 127.0.0.1) を特定します。その他の設定はそのままにします。

```
KUBE_MASTER="--master=http://master.example.com:8080"
```

3. **/etc/kubernetes/kubelet の編集**: 各ノードにあるこのファイルで、以下のように KUBELET_ADDRESS (すべてのネットワークインターフェースで listen するように 0.0.0.0)、KUBELET_HOSTNAME (hostname_override をローカルシステム node1 または node2 のホスト名または IP アドレスに置き換える) を変更し、KUBELET_ARTS を "--register-node=true" に設定し、KUBELET_API_SERVER を設定します (--api_servers=http://master.example.com:8080 またはマスターの他の場所に設定)。

```
KUBELET_ADDRESS="--address=0.0.0.0"
KUBELET_HOSTNAME="--hostname-override=node?"
KUBELET_ARGS="--register-node=true"
KUBELET_API_SERVER="--api_servers=http://master.example.com:8080"
```

4. **/etc/kubernetes/proxy の編集**: このファイルを設定する必要はありません。KUBE_PROXY_ARGS が設定されている場合、これをコメントアウトすることができます。

```
# KUBE_PROXY_ARGS="--master=http://master.example.com:8080"
```

5. **Kubernetes Node の systemdサービスの起動**: 各ノードで、Kubernetes Node と関連付けられた複数のサービスを起動する必要があります。

```
# for SERVICES in docker kube-proxy.service kubelet.service; do
    systemctl restart $SERVICES
    systemctl enable $SERVICES
    systemctl status $SERVICES
done
```

6. **サービスの確認**: 3 つのシステムのそれぞれに対して **netstat** コマンドを実行し、サービスが実行されているポートを確認します。etcd サービスはマスターでのみ実行されます。

   - マスター:

```
# netstat -tulnp | grep -E "(kube)|(etcd)"
```

   - ノード:

```
# netstat -tulnp | grep kube
```

   注意: この手順では firewalld サービスが Atomic にインストールされていません (また

RHEL で無効しておく必要があります) が、システム上のポートへのアクセスをブロックするために iptables ルールを直接使用することもできます。それを使用した後は、この **netstat** コマンドを実行すると、Kubernetes クラスターを機能させるために各システムでアクセス可能な状態にしておく必要のあるポートが表示されます。そのため、個別のファイアウォールルールを作成して、これらのポートへのアクセスを開くことができます。

7. **etcd サービスのテスト**: これらの 3 つのシステムのいずれかから curl コマンドを実行して、etcd サービスが実行中でアクセス可能であることを確認します。

```
# curl -s -L http://master.example.com:2379/version
{"etcdserver":"2.2.0","etcdcluster":"2.2.0"}
```

8. **ノードの確認**: マスターから以下のコマンドを入力して、マスターが 2 つのノードと通信していることを確認します。

```
# kubectl get nodes
NAME              LABELS
STATUS
node1.example.com kubernetes.io/hostname=node1.example.com Ready
node2.example.com kubernetes.io/hostname=node2.example.com Ready
```

これで、Kubernetes クラスターが設定されました。次にこの設定を使用してネットワークを設定し、Kubernetes サービス、レプリケーションコントローラーおよび Pod をデプロイして、コンテナーのデプロイを開始することができます。

3.6. KUBERNETES の FLANNEL ネットワークのセットアップ

flannel パッケージには、Kubernetes クラスターのマスターとノード間のネットワークを設定できる機能が含まれています。flanneled サービスは、(マスターで) etcd サーバーへのネットワーク設定を含んだ json 設定ファイルを作成し、アップデートして設定します。次に flanneld systemd サービスをマスターとおよび各ノードに設定してその etcd サーバーを指定し、flanneld サービスを起動します。この手順では、その方法を説明します。

注意: etcd サービスは、flanneld が起動する前に実行されている必要があります。flanneld および docker がノードで起動しない場合は、etcd がマスターで稼働していることを確認し、各ノードを再起動するようにしてください。

この例では、flannel は kubernetes 環境のすべてのノードで使用されるアドレス範囲 **10.20.0.0/16** を割り当てます。これにより、マスターと両ノードのネットワークインターフェースに割り当てられる範囲内に別個の /24 サブネットを使用できます。この例の 3 つのシステムの場合、以下の手順を実行すると、以下のアドレス範囲が flannel.1 および docker0 インターフェースに割り当てられます。

- マスター:
  - flannel.1: 10.20.21.0/16
- node1:
  - flannel.1: 10.20.26.0/16
  - docker0: 10.20.26.1/24
- node2:
  - flannel.1: 10.20.37.0/16

- docker0: 10.20.37.1/24

重要: docker0 インターフェースは、この手順の実行時にはすでに配置されているため、flanneld により docker0 に割り当てられる IP アドレス範囲はすぐには有効になりません。flanneld アドレス範囲を有効にするには、docker を停止 (**systemctl stop docker**) して docker0 インターフェースを削除し、docker インターフェースを再起動 (**systemctl start docker**) する必要があります。この手順の実行時には、単に再起動するよう指示されます。

1. **flannel のインストール**: flannel パッケージは RHEL Atomic にプリインストールされています。ただし RHEL 7 を使用している場合は、以下のようにマスターと各ノードに flannel パッケージをインストールできます。

```
# yum install flannel
```

2. **flannel 設定ファイルの作成**: マスターで、flannel 設定ファイル (json 形式) の IP アドレスのセットおよびネットワークの種類を指定します。この例では、以下の内容を含む **flannel-config.json** というファイルを作成しています。

```
{
  "Network": "10.20.0.0/16",
  "SubnetLen": 24,
  "Backend": {
    "Type": "vxlan",
    "VNI": 1
  }
}
```

3. **flannel 設定の etcd サービスへのアップロード**: マスターで flannel 設定ファイルを etcd サービスにアップロードするには、以下を入力します。

```
# etcdctl set coreos.com/network/config < flannel-config.json
  {"action":"set","node":
{"key":"/coreos.com/network/config","value":"{\n\"Network\":
\"10.20.0.0/16\",\n\"SubnetLen\": 24,\n\"Backend\": {\n\"Type\":
\"vxlan\",\n\"VNI\": 1\n
}\n}\n","modifiedIndex":10,"createdIndex":10},"prevNode":
{"key":"/coreos.com/network/config","value":"","modifiedIndex":9,
"createdIndex":9}}
```

次に、アップロードが適切に完了したことを確認します。

```
# etcdctl get coreos.com/network/config
{
  "Network": "10.20.0.0/16",
  "SubnetLen": 24,
  "Backend": {
    "Type": "vxlan",
    "VNI": 1
  }
}
```

4. **flanneld のオーバーレイネットワークの設定**: flanneld systemd サービスが起動すると、オプションの **/etc/sysconfig/flanneld** ファイルを読み込んで、flanneld デーモンに渡します。マスターおよび両ノードで **/etc/sysconfig/flanneld** を編集して、etcd サービス (マスター) を含むシステムの名前または IP アドレス名を追加します。FLANNEL_ETCD_KEY

行はそのままにします。

```
FLANNEL_ETCD="http://master.example.com:2379"
FLANNEL_ETCD_KEY="/coreos.com/network"
```

5. **マスターとノードでの flanneld の起動**: 以下のように最初にマスター、次に 2 つのノードで flanneld サービスを起動し、有効にします。

```
# systemctl restart flanneld
# systemctl enable flanneld
# systemctl status flanneld
```

6. **flannel.1 ネットワークインターフェースの確認**: 任意のシステムから以下のコマンドを実行して、flannel.1 ネットワークインターフェースが各システム上に適切に設定されていることを確認します。最初のコマンドは、各ノードで ip a コマンドを実行し、flannel.1 インターフェースを確認します。2 つ目のコマンドは、サブネットが各システムの flannel に対して設定されている内容を示します。

```
# for i in 1 2; do ssh root@node$i ip a l flannel.1; done | grep
'inet '
root@node1's password: password
    inet 10.20.26.0/16 scope global flannel.1
root@node2's password: password
    inet 10.20.37.0/16 scope global flannel.1
# for i in master node1 node2; \
     do echo --- $i ---; ssh root@$i cat
/run/flannel/subnet.env; done
root@master's password: password
--- master ---
FLANNEL_SUBNET=10.20.21.1/24
FLANNEL_MTU=1450
root@node1's password: password
--- node1 ---
FLANNEL_SUBNET=10.20.26.1/24
FLANNEL_MTU=1450
root@node2's password: password
--- node2 ---
FLANNEL_SUBNET=10.20.37.1/24
FLANNEL_MTU=1450
```

7. **ノードの再起動**: docker の systemd サービスが flannel の変更を選択するようにし、すべてのネットワークインターフェースが適切に起動することを確認するには、各ノードで以下を実行して、再起動します。

```
# systemctl reboot
```

8. **flannel アドレスを取得するためのコンテナーの起動**: 各ノードで flannel ネットワークのアドレス範囲を確認し、ネットワークからアドレスが選択されることを確認するためにイメージを実行します。

```
# ip a | grep flannel
3: flannel.1:  mtu 1450 qdisc noqueue state UNKNOWN
inet 10.20.26.0/16 scope global flannel.1
# docker run -d --name=mydbforweb dbforweb
# docker inspect --format='{{.NetworkSettings.IPAddress}}'
```

```
mydbforweb
10.20.26.2
# docker stop mydbforweb
# docker rm mydbforweb
```

3.7. KUBERNETES によるサービス、レプリケーションコントローラー、およびコンテナー POD の起動

Kubernetes クラスターを設定したら、Kubernetes サービスをセットアップするために必要な yaml ファイルを作成し、レプリケーションコントローラーを定義し、コンテナーの Pod を起動できます。前述の 2 つのコンテナー (Web および DB) を使用して、以下の種類の Kubernetes オブジェクトを作成します。

- **サービス**: Kubernetes サービスを作成すると、特定の IP アドレスおよびポート番号をラベルに割り当てることができます。Pod および IP アドレスは Kubernetes と連携するため、そのラベルを必要なサービスの場所を特定するために Pod 内で使用できます。
- **レプリケーションコントローラー**: レプリケーションコントローラーを定義すると、起動する Pod を設定するだけではなく、起動する各 Pod のレプリカの数を設定することができます。Pod が停止すると、レプリケーションコントローラーは別の Pod を起動し、これを置き換えます。
- **Pod**: Pod は 1 つ以上のコンテナーを、コンテナーの実行に関係したオプションと共にロードします。

この例では、2 つの Pod が相互に通信できるようにするために必要なサービスオブジェクト (yaml ファイル) を作成します。次に、前述の Web サーバーおよびデータベースサーバーのコンテナーを起動し、維持するための Pod を特定するレプリケーションコントローラーを作成します。本書で使用されるすべての yaml および json 設定ファイルを取得するには、以下の tarball をダウンロードします。

kube_files.tar

1. **データベースサーバー用の Kubernetes サービスのデプロイ**:マスター上で、データベースサーバーコンテナーを特定の IP アドレスとポートに関連付けるためにラベルを指定する Kubernetes サービスの yaml ファイルを作成します。以下は、**db-service.yaml** というファイルを使用した例です。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: db
  name: db-service
  namespace: default
spec:
  ports:
  - port: 3306
  selector:
    name: db
```

このサービスは外部から直接アクセスされません。webserver コンテナーがこれにアクセスします。セレクターおよびラベルの名前を db に設定します。サービスを起動するには、マスターで以下を入力します。

```
# kubectl create -f db-service.yaml
service "db-service" created
```

2. **データベースサーバー用の Kubernetes ReplicationController のデプロイ**: 実行しているデータベース Pod の数 (この場合は 2 つ) を指定するレプリケーションコントローラーの yaml ファイルを作成します。この例では、ファイルの名前は **db-rc.yaml** です。

```
kind: ReplicationController
apiVersion: v1
metadata:
   name: db-controller
spec:
  replicas: 2
  selector:
    name: db
  template:
    spec:
      containers:
      - name: db
        image: dbforweb
        ports:
          - containerPort: 3306
    metadata:
      labels:
        name: "db"
      selectorname: "db"
  labels:
    name: "db"
```

webserver pod のレプリケーションコントローラーを起動するには、以下を入力します。

```
# kubectl create -f db-rc.yaml
replicationcontroller "db-controller" created
```

3. **Web サーバー用の Kubernetes サービスの yaml ファイルの作成とその起動**: マスター上で Kubernetes サービスの yaml ファイルを作成します。このファイルは、Web サーバーのコンテナーを特定の IP アドレスおよびポートに関連付けるラベルを指定します。以下は、**webserver-service.yaml** ファイルを使用した例です。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: webserver
  name: webserver-service
  namespace: default
spec:
  ports:
  - port: 80
  publicIPs:
  - 192.168.122.15
  selector:
    name: webserver
```

porrtalIP は、**/etc/kubernetes/apiserver** ファイルで設定される範囲内にある必要がありま

す (この場合は 10.254.100.0/24)。publicIPs 値は、ノードのいずれかにある外部ネットワークに関連付けられた IP アドレスと連携する必要があります (node2 では eth0 インターフェースの IP アドレスは 192.168.122.15 です)。これにより、サービスが外部にも利用可能になります。セレクターおよびラベルの名前は webserver に設定します。サービスを起動するには、マスターで以下を入力します。

```
# kubectl create -f webserver-service.yaml
service "webserver-service" created
```

apiserver および追加したばかりの 2 つのサービスが実行中の状態で、以下のコマンドを入力し、それらのサービスを表示します。

```
# kubectl get services
NAME              LABELS
SELECTOR
          IP(S)          PORT(S)
db-service        name=db
name=db
          10.254.192.67  3306/TCP
kubernetes        component=apiserver,provider=kubernetes
          10.254.255.128 443/TCP
kubernetes-ro     component=apiserver,provider=kubernetes
          10.254.139.205 80/TCP
webserver-service name=webserver
name=webserver
          10.254.134.105 80/TCP
```

4. **Web サーバー用の Kubernetes ReplicationController のデプロイ**: 実行している webserver Pod の数を特定するレプリケーションコントローラーの yaml ファイルを作成します。以下の yaml ファイルに基づいて、レプリケーションコントローラーは「webserver」ラベルの付いた 2 つの Pod を常に実行するようにします。Pod 定義はレプリケーションコントローラーの yaml ファイル内にあるため、別の Pod の yaml ファイルは不要です。以下は、**webserver-rc.yaml** というファイルの例です。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: webserver-controller
spec:
  replicas: 2
  selector:
    name: webserver
  template:
    spec:
      containers:
        - name: apache-frontend
          image: webwithdb
          ports:
            - containerPort: 80
    metadata:
      labels:
        name: webserver
        uses: db
  labels:
    name: "webserver"
```

webserver pod のレプリケーションコントローラーを起動するには、以下を入力します。

```
# kubectl create -f webserver-rc.yaml
replicationcontroller "webserver-controller" created
```

3.8. KUBERNETES の確認

アプリケーションが機能していることを確認するには、2 つの **curl** コマンドを実行して、Web サーバーからデータを取得することができます。クラスターが適切に機能していることを確認するには、いくつかの **kubectl** コマンドを実行できます。

1. **Web サーバーの確認**: 以下のコマンドを実行して、Web サーバーからデータを取得します。Pod アドレスを使用して web サービスにアクセスします。

```
# kubectl get endpoints | grep web
NAME                ENDPOINTS                          AGE
webserver-service   10.20.11.3:80,10.20.55.2:80        3d
# curl http://10.20.11.3/index.html
The Web Server is Running
# curl http://10.20.11.3/cgi-bin/action
<html>
<head>
<title>My Application</title>
</head>
<body>
<h2>RedHat rocks</h2>
<h2>Success</h2>
</body>
</html>
```

2. **Kubernetes クラスターの確認**: マスターから以下のいくつかのコマンドを実行すると、クラスターを確認できます。

```
# kubectl get pods
# kubectl get replicationControllers
# kubectl get services
# kubectl get nodes
```

3. **Kubernetes Node の確認およびコンテナーの一時停止**: Pod が長い間保留状態になっている場合は、Pod が起動しようとしているコンテナーを選択したノードで起動できないことが原因である場合があります。特定のノード上で実行されているコンテナーをチェックアウトするには、そのノードにログインしてから **docker ps** コマンドを入力します。

```
# docker ps
CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS
PORTS               NAMES
b28a863e0aca        dbforweb
"/usr/bin/mysqld_safe"   2 hours ago         Up 2 hours
k8s_db...
b138c90519a1        webwithdb
"/usr/sbin/httpd -D F"   2 hours ago         Up 2 hours
k8s_apache...
872c5acf3ec8        beta.gcr.io/google_containers/pause:2.0
```

```
"/pause"                2 hours ago         Up 2 hours
k8s_POD...
e117af93e4f4        beta.gcr.io/google_containers/pause:2.0
"/pause"                2 hours ago         Up 2 hours
k8s_POD...
```

上記の出力は、マスターから 2 つのレプリケーションコントローラーが正常に起動されたことを示しています。dbforweb および webwithdb コンテナーはこのノード上で実行されています。ただし、コマンド **/pause** を実行している他の 2 つのコンテナーも実行中のコンテナーの一覧に表示されていることに注意してください。

pause コンテナーは特殊なタイプのコンテナーです。ここに示されるコンテナーはアップストリームの Kubernetes プロジェクトから取られています。各 Pod について、pause コンテナーは Pod 内の最後のネットワーク名前空間をノード上で予約するために使用され、Pod が行き来する際に名前空間が確保されるようにします。デフォルトのすべての **pause** コンテナーが実行するアクションは休止 (sleep) のみです。このコンテナーは Kubernetes により自動的に取得されます。

pause コンテナーに関する潜在的な問題とは、取得元となるレジストリーから切断された状態で Kubernetes クラスターを実行する必要がある場合に、必ず **pause** コンテナーを取得してから接続を切断する必要がある点にあります。

直前の例で使用されたデフォルトのコンテナーの代わりとして他の **pause** コンテナーを使用することもできます。正式な Red Hat の pause コンテナーは **rhel7/pod-infrastructure** と呼ばれており、まもなく利用可能になります。

3.9. KUBERNETES の削除

クラスターの使用が終了したら、適切に削除できるように特定の方法で削除する必要があります。

ここで、順序は重要になります。replicationController を削除する前に Pod を削除すると、Pod は再起動します。以下のようになります。

```
# kubectl delete replicationControllers webserver-controller
# kubectl delete replicationControllers db-controller
# kubectl delete services webserver-service
# kubectl delete services db-service
# kubectl get pods
POD                                    IP          CONTAINER(S)    ...
89d8577d-bceb-11e4-91c4-525400a6f1a0 10.20.26.4  apache-frontend ...
8d30f2de-bceb-11e4-91c4-525400a6f1a0 10.20.26.5  db              ...
# kubectl delete pods 89d8577d-bceb-11e4-91c4-525400a6f1a0
# kubectl delete pods 8d30f2de-bceb-11e4-91c4-525400a6f1a0
```

Kubernetes クラスターを継続して使用する場合は、各ノードで以下を実行して、サービスが稼働していることを確認します。

```
# for SERVICES in docker kube-proxy.service kubelet.service; do
    systemctl restart $SERVICES
    systemctl status $SERVICES
    done
```

ノードの使用を終了した場合は、以下のようにマスターから削除できます。

```
# kubectl delete node node1.example.com
# kubectl delete node node2.example.com
```

3.10. 添付ファイル

kube_files.tar

第4章 KUBERNETES のトラブルシューティング

4.1. 概要

Kubernetes は、Docker フォーマットコンテナーのセットに、アプリケーションをデプロイし、管理するための機能セットを提供します。本書では、Kubernetes Pod、レプリケーションコントローラー、サービス、および結果として生成されるコンテナーの作成および管理に関連した問題のトラブルシューティングに役立ちます。

トラブルシューティングの手法を説明するために、本書では「Kubernetes クラスターを作成して Docker フォーマットのコンテナーイメージを実行する」でデプロイされているコンテナーおよび設定を使用します。ここで説明されている手法は、Red Hat Enterprise Linux 7.1 および RHEL Atomic Host システムで実行される Kubernetes に適用されます。

4.2. KUBERNETES のトラブルシューティングについて

Kubernetes のトラブルシューティングを開始する前に、調査対象となる Kubernetes コンポーネントを理解しておく必要があります。それらには以下が含まれます。

- **Master**: Kubernetes 環境を管理するために使用するシステムです。
- **Node**: コンテナーが Kubernetes によってデプロイされる 1 つ以上のシステムです (以前は minion と呼ばれていました)。
- **Pod**: Pod は、各コンテナーに対する docker run コマンドへのオプション、サービスの位置を定義するためのラベルと共に、実行する 1 つ以上のコンテナーを定義します。
- **サービス**: サービスにより、Kubernetes 環境内のコンテナーが IP アドレスを識別しなくても、別のコンテナーが提供するアプリケーションを名前 (ラベル) で見つけられるようになります。
- **レプリケーションコントローラー**: レプリケーションコントローラーを使用すると、特定の数の Pod が実行されるように指定できます (新規の Pod は必要数に達するまで起動した状態となり、Pod が終了すると、新規の Pod がこれを置き換えるために実行されます。)
- **ネットワーキング (flanneld)**: flanneld サービスを使用すると、Kubernetes が使用する IP アドレス範囲の設定と関連の設定を行うことができます。この機能はオプションです。yaml および json ファイルが使用されます。使用する Kubernetes の要素は、yaml または json 形式の設定ファイルで作成されます。本書では主に yaml 形式のファイルに焦点を当てます。

とくに以下のコマンドを使用して、上記のコンポーネントのトラブルシューティングを行います。

- **kubectl**: (マスターから実行される) **kubectl** コマンドにより、Kubernetes Pod, サービスおよびレプリケーションコントローラーで、作成、削除、取得 (情報の一覧表示) などの操作を実行できます。このコマンドを使用して yaml/json ファイルをテストし、各種の Kubernetes コンポーネントの状態を表示できます。
- **systemctl**: マスターとノード間の通信を容易にするために特定の **systemd** サービスを Kubernetes で設定する必要があります。さらに、それらのサービスはアクティブかつ有効な状態にしておく必要があります。
- **journalctl**: **journalctl** コマンドを使用して Kubernetes の systemd サービスを確認し、これらのサービスの処理に従います。これをマスターとノードの両方で実行し、それらのシステム上での Kubernetes の障害があるかどうかを確認します。kubernetes でのすべてのデーモンロギングでは systemd ジャーナルを使用します。

- **etcdctl** または **curl**: **etcd** デーモンは、Kubernetes クラスター情報の保管を管理します。このサービスは、マスターまたはその他のシステムで実行できます。その情報を照会するには RHEL の etcdctl コマンドを使用することができます。RHEL Atomic には etcdctl コマンドがないため、代わりに curl コマンドを使用して etcd サービスを照会できます。

4.3. KUBERNETES 用にコンテナー化されたアプリケーションの準備

アプリケーションを kubernetes にデプロイする前に考慮すべきいくつかの点について以下に説明します。

4.3.1. ネットワークの制限

Kubernetes サービスとして実行できるアプリケーションの種類には制限があるため、すべてのアプリケーションを一律に kubernetes に対応させることはできません。Kubernetes におけるサービスは、IP アドレスがサービスのクライアントの iptables に挿入される負荷分散プロキシーです。したがって、Kubernetes にデプロイする予定のアプリケーションについては、以下に該当するかを確認する必要があります。

- サブプロセス間でのネットワークアドレス変換や NAT をサポートする。
- フォワードおよびリバース DNS ルックアップを必要としない。Kubernetes はサービスのクライアントにフォワードまたはリバース DNS ルックアップを提供しません。

上記のいずれかに関連する制限が適用されないか、またはユーザーがそれらの確認項目を無効にできる場合は、このまま継続することができます。

4.3.2. コンテナーの準備

実行しているソフトウェアの種類によって、Kubernetes サービスのクライアントに提供される事前に定義された一部の環境変数を利用できる場合があります。

たとえば、**db** という名前のサービスの場合、そのサービスを使用する Kubernetes で Pod を起動する場合、Kubernetes は以下の環境変数を Pod のコンテナーに挿入します。

```
DB_SERVICE_PORT_3306_TCP_PORT=3306
DB_SERVICE_SERVICE_HOST=10.254.100.1
DB_SERVICE_PORT_3306_TCP_PROTO=tcp
DB_SERVICE_PORT_3306_TCP_ADDR=10.254.100.1
DB_SERVICE_PORT_3306_TCP=tcp://10.254.100.1:3306
DB_SERVICE_PORT=tcp://10.254.100.1:3306
DB_SERVICE_SERVICE_PORT=3306
```

注意: サービス名 (db) は、変数では DB のように大文字で表示されることに注意してください。名前に使用されるダッシュ (-) は、アンダースコア (_) に変換されます。

これらの変数を含むシェル変数を確認するには、**docker exec** を実行してアクティブなコンテナーにシェルを開き、**env** を実行してシェル変数を確認します。

```
# docker exec -it <container_ID> /bin/bash
[root@e7ea67..]# env
...
WEBSERVER_SERVICE_SERVICE_PORT=80
KUBERNETES_RO_SERVICE_PORT=80
KUBERNETES_SERVICE_PORT=443
KUBERNETES_RO_PORT_80_TCP_PORT=80
```

```
KUBERNETES_SERVICE_HOST=10.254.255.128
DB_SERVICE_PORT_3306_TCP_PORT=3306
DB_SERVICE_SERVICE_HOST=10.254.100.1
WEBSERVER_SERVICE_PORT_80_TCP_ADDR=10.254.100.50
...
```

クライアントアプリケーションを起動する際に、これらの変数を利用する必要がある場合があります。たとえば、「Kubernetes クラスターを作成して Docker フォーマットのコンテナーイメージを実行する」で説明されているように Web サーバーや Database アプリケーションを作成する場合、Web サーバーがデプロイする action スクリプトは DB_SERVICE_SERVICE_HOST の値を取得するため、データベースサービスにアクセスするために使用できる IP アドレスを認識できます。

コンテナー間の通信の問題をデバッグする際にこれらのシェル変数を表示すると、サービスおよびポートのアドレスをコンテナーごとに表示するのに役立ちます。

4.4. KUBERNETES のデバッグ

Kubernetes のデバッグを開始する前に、Kubernetes の機能の概要を理解しておく必要があります。通常、アプリケーションを Kubernetes に送る際に、以下が実行されます。

1. 認証時に kubectl コマンドラインが (マスターの) kube-apiserver に送られる。
2. (マスターの) kube-scheduler プロセスが yaml または json ファイルを読み取り、Pod をノードに割り当てる (ノードは kubelet サービスを実行するシステム)。
3. (ノードの) kublet サービスは、Pod のマニフェストを 1 つ以上の **docker run** 呼び出しに変換する。
4. **docker run** コマンドは、利用可能なノードで特定されたコンテナーの起動を試行する。

そのため、kubernetes でデプロイされたアプリケーションをデバッグするには、以下を確認する必要があります。

1. Kubernetes サービスのデーモン (systemd) プロセスが実行中である。
2. yaml または json の送信が有効である。
3. kubelet サービスが kube-scheduler から作業の順番を受信する。
4. 各ノードの kubelet サービスは docker を使用した各コンテナーを正常に起動できる。

注記

上記の一覧には、レプリケーションコントローラーを作成する場合などに重要となる kube-controller-manager がありませんが、これに管理されている Pod がないことを確認できます。またノードをクラスターで登録していますが、利用可能なリソースなどの情報は取得されません。

さらに、アップストリームから一体型の hyperkube バイナリーに移動したため、ここで使用されている用語は近い将来に変更される可能性があります。

4.4.1. Kubernetes の検査およびデバッグ

Kubernetes Master から、実行中の Kubernetes 設定を検査します。上述したように、このセクションのトラブルシューティングの例は、「Kubernetes クラスターを作成して Docker フォーマットのコンテナーイメージを実行する」に記載されている設定で実行されます。

まず、すべての設定を有効にした場合どのように表示されるかについて説明します。次に、セットアップに発生する各種の問題とその修正方法を示します。

4.4.2. Kubernetes の状態の照会

アプリケーションの送信とサービスの作成、 Pod の割り当てのプロセスを手動でデバッグするには、**kubectl** を使用するのが一番簡単です。どの Pod、サービス、およびレプリケーションコントローラーがアクティブであるかを確認するには、マスターで以下のコマンドを実行します。

```
# kubectl get pods
POD             IP          CONTAINER(S)     IMAGE(S)  HOST
LABELS                                         STATUS
4e04dd3b-c...   10.20.29.3  apache-frontend  webwithdb
node2.example.com/  name=webserver,selectorname=webserver,uses=db
Running
5544eab2-c...   10.20.48.15 apache-frontend  webwithdb
node1.example.com/  name=webserver,selectorname=webserver,uses=db
Running
1c971a09-c...   10.20.29.2  db               dbforweb
node2.example.com   name=db,selectorname=db
Running
1c97a755-c...   10.20.48.14 db               dbforweb
node1.example.com/  name=db,selectorname=db
Running
# kubectl get services
NAME               LABELS                                    SELECTOR
IP                  PORT
webserver-service  name=webserver
name=webserver  10.254.100.50       80
db-service         name=db                                   name=db
10.254.100.1        3306
kubernetes         component=apiserver,provider=kubernetes
10.254.92.19        443
kubernetes-ro      component=apiserver,provider=kubernetes
10.254.206.141      80
# kubectl get replicationControllers
CONTROLLER             CONTAINER(S)        IMAGE(S)
SELECTOR                 REPLICAS
webserver-controller   apache-frontend     webwithdb
selectorname=webserver   2
db-controller          db                  dbforweb
selectorname=db          2
```

以下に、この出力が何を示しているかを説明します。

- Pod の状態は「Waiting」または「Running」です。ここでは 4 つの Pod がすべて実行されているため、問題はありません。
- レプリケーションコントローラーは、2 つの apache-frontend と 2 つの db コンテナーを正常に起動しました。それらは node1 および node2 に配信されます。

- apache-frontend の **uses** ラベルを使用して、コンテナーが db-service Kubernetes サービスから db コンテナーを見つけられるようにします。
- サービスの一覧は、各サービスのラベル名で、Pod から要求できる各サービスの IP アドレスとポート番号を特定します。
- kubernetes および kubernetes-ro サービスは、kube-apiserver systemd サービスへのアクセスを提供します。

この状態になるまでのプロセスで問題が発生した場合は、以下のセクションを参照してトラブルシューティングを行うことができます。

4.5. KUBERNETES SYSTEMD サービスのトラブルシューティング

Kubernetes は、Kubernetes のマスターおよびノードで実行されるサービスデーモンのセットを使用して実装されます。それらの systemd サービスが適切に実行されていないと失敗します。Kubernetes systemd サービスを使用した場合に潜在的な問題を回避し、それらを修正する方法を以下に示します。

4.5.1. Kubernetes systemd サービスが起動していることの確認

Kubernetes クラスターは、マスターと 1 つ以上のノード (minion) で構成されており、systemd サービスの特定セットを初期化する必要があります。マスターおよび各ノードで以下のサービスが実行していることを確認する必要があります。

- **マスターを最初に起動する**: ノードのサービスを起動する前にマスターのサービスを起動する必要があります。マスターが起動していない場合はノードは適切に起動されません。
- **マスターサービス**: サービスには、kube-controller-manager、kube-scheduler、flanneld、etcd、および kube-apiserver が含まれます。flanneld サービスはオプションのサービスで、別のシステムで etcd サービスを実行することができます。
- **ノードサービス**: サービスには、docker、kube-proxy、kubelet、flanneld が含まれます。flanneld サービスはオプションです。

以下は、マスターおよび各ノード上でこれらのサービスを確認する方法です。

マスター: kubernetes マスターサーバーでは、適切なサービスがアクティブであり、有効にされているかどうかが示されます (flanneld はシステムに設定されていない場合もあります)。

```
# for SERVICES in etcd flanneld kube-apiserver kube-controller-manager
kube-scheduler;
    do echo --- $SERVICES --- ; systemctl is-active $SERVICES ;
    systemctl is-enabled $SERVICES ; echo "";  done
--- etcd ---
active
enabled
--- flanneld ---
active
enabled
--- kube-apiserver ---
active
enabled
--- kube-controller-manager ---
active
```

```
enabled
--- kube-scheduler ---
active
enabled
```

Minions (ノード): 各ノードで、適切なサービスがアクティブであり、有効にされていることを確認します。

```
# for SERVICES in flanneld docker kube-proxy.service kubelet.service; \
do echo --- $SERVICES --- ; systemctl is-active $SERVICES ; \
systemctl is-enabled $SERVICES ; echo "";  done
--- flanneld ---
active
enabled
--- docker ---
active
enabled
--- kube-proxy.service ---
active
enabled
--- kubelet.service ---
active
enabled
```

マスターまたはノードの systemd サービスのいずれかが無効にされているか、またはこれに障害がある場合は、以下を実行します。

- Kubernetes クラスターのセクションの Kubernetes サービスの起動について説明している部分を参照して、サービスの有効化またはアクティブ化を試行します。
- サービスに障害が発生しているシステムで systemd ジャーナルを確認し、問題解決のヒントを探します。1 つの方法として、サービスを表示するコマンドで journalctl を使用することができます。以下は例になります。

  ```
  # journalctl -l -u kubelet
  # journalctl -l -u kube-apiserver
  ```

- サービスが依然として起動しない場合は、Kubernetes クラスターの部分を再度参照し、各サービスの設定ファイルが適切にセットアップされていることを確認します。

4.5.2. Kubernetes のファイアウォールの確認

RHEL Atomic には iptables または firewalld サービスはインストールされていません。そのため、デフォルトでは Kubernetes サービスへのアクセスをブロックするファイアウォールのフィルタールールはありません。ただし、RHEL ホストでファイアウォールが実行している場合や、Kubernetes マスターまたはノードに iptables ファイアウォールルールを追加して入力ポートにフィルターを設定している場合は、これらのシステムに公開する必要があるポートがブロックされていないことを確認する必要があります。

以下は **netstat** コマンドの出力であり、Kubernetes および関連サービスが Kubernetes ノードで listen しているポートを示しています。

```
# netstat -tupln
tcp6       0      0 :::10249                :::*                    LISTEN
125528/kube-proxy
tcp6       0      0 :::10250                :::*                    LISTEN
```

```
125536/kubelet
```

注意: kube-proxy サービスはランダムなポートで listen します。デフォルトで使用されているファイアウォールサービスにフィルタリングが使用されているため、これは RHEL Atomic システムでは問題ありません。ただし、ファイアウォールを RHEL Atomic に追加するか、またはデフォルトの RHEL システムを使用する場合、サービス定義で kube-proxy が特定のポートで listen し、ファイアウォールでそれらのポートを開くように設定することができます。

以下は、Kubernetes マスターにおける **netstat** 出力です。

```
tcp        0      0 192.168.122.249:7080  0.0.0.0:* LISTEN   636/kube-
apiserver
tcp6       0      0 :::8080               :::*      LISTEN
636/kube-apiserver
tcp        0      0 127.0.0.1:10252       0.0.0.0:* LISTEN
7541/kube-controller
tcp        0      0 127.0.0.1:10251       0.0.0.0:* LISTEN
7590/kube-scheduler
tcp6       0      0 :::4001               :::*      LISTEN   941/etcd
tcp6       0      0 :::7001               :::*      LISTEN   941/etcd
```

出力の 3 列目を参照してください。

ここでは、各サービスが listen している IP アドレスとポート番号が表示されています。それらの各サービスに対してポートが開かれていることを確認します。

4.5.3. Kubernetes の yaml または json ファイルの確認

kubectl create コマンドを使用して、yaml または json ファイルから情報をロードし、Kubernetes 環境 (Pod、サービス、およびレプリケーションコントローラー) をセットアップします。それらのファイルが正しくフォーマットされていなかったり、必要な情報が不足していると失敗します。

以下のセクションでは、yaml または json ファイルが破損している場合に生じる問題を解決するためのヒントを紹介します。

4.5.3.1. Kubernetes サービス作成時のトラブルシューティング

Kubernetes サービス (**kubectl** で作成) は IP アドレスおよびポートをラベルに割り当てます。このサービスを使用する必要のある Pod はラベルでそのサービスを参照できるため、IP アドレスおよびポート番号を直接認識する必要はありません。以下は、db-service.yaml という名前のサービスファイルの例とサービスを作成する際に生じる問題の一覧です。

```
id: "db-service"
kind: "Service"
apiVersion: "v1beta1"
port: 3306
portalIP: "10.254.100.1"
selector:
  name: "db"
labels:
  name: "db"
```

```
# kubectl create -f db-service
# kubectl get services
```

```
NAME          LABELS                                    SELECTOR  IP
PORT
db-service    name=db                                   name=db
10.254.100.1   3306
kubernetes-ro component=apiserver,provider=kubernetes     10.254.186.33
80
kubernetes    component=apiserver,provider=kubernetes     10.254.198.9
443
```

注意: kubernetes-ro および kubernetes サービスが表示されない場合は、kube-scheduler systemd サービス (**systemctl restart kube-scheduler.service**) を再起動します。

表示された結果がこれと異なる場合は、以下を参照してください。

- サービスが正常に作成されている場合でも、LABELS と SELECTOR が設定されていない場合、出力結果は以下のようになる可能性があります。

  ```
  # kubectl get services
  NAME         LABELS     SELECTOR  IP             PORT
  db-service            10.254.100.1   3306
  ```

  selector: および labels: の下の name: フィールドがそれぞれ 2 文字分インデントされていることを確認します。この場合、各 **name: "db"** 行の前にある 2 つの空白を削除したので、それらの値は **kubectl** で使用されませんでした。

- サービスを作成したことを忘れて再作成しようとしたり、サービスの yaml で特定した IP アドレスに他のサービスがすでに割り当てられている場合でサービスを新たに作成しようとすると、以下のメッセージが生成されます。

  ```
  create.go:75] service "webserver-service" is invalid: spec.portalIP:
      invalid value '10.254.100.50': IP 10.254.100.50 is already
  allocated
  ```

  そのサービスが必要ない場合は、異なる IP アドレスを使用するか、現在そのポートを使用しているサービスを停止するかのいずれかを実行できます。

- "Service" オブジェクトが登録されていないことを示す以下のようなエラーが発生する理由がいくつかあります。

  ```
  7338 create.go:75] unable to recognize "db-service.yaml": no object
  named "Services" is registered
  ```

  上記の例では、「Service」は「Services」と誤って綴られています。「Service」と正しく綴られている場合は、apiVersion が正しいことを確認します。同様のエラーは、無効な値 "v1beta99" が apiVersion に使用される場合に生じます。その場合、"v1beta99" が存在しないことではなく、オブジェクトの「Service」が見つからないことを報告します。

    - 以下は、上記の例のフィールドがない場合に発生するエラーメッセージの一覧です。

      - **id: missing**: service "" is invalid: name: required value ''
      - **kind: missing**: unable to recognize "db-service.yaml": no object named "" is registered
      - **apiVersion: missing**: service "" is invalid: [name: required value '', spec.port: invalid value '0']

- **port: missing**: service "db-service" is invalid: spec.port: invalid value '0'
- **portalIP: missing**: No error is reported because portalIP is not required
- **selector: missing**: No error is reported, but SELECTOR field is missing and service may not work.
- **labels: missing**: Not an error, but LABELS field is missing and service may not work.

4.5.3.2. Kubernetes レプリケーションコントローラーおよび Pod 作成のトラブルシューティング

Kubernetes の Pod を使用して、1 つ以上のコンテナーを関連付け、実行オプションを各コンテナーに割り当て、それらのコンテナーをまとめてユニットとして管理します。レプリケーションコントローラーを使用すると、確認する Pod をいくつ実行するかを指定できます。以下は、Web サーバーの Pod を定義する yaml ファイルと、2 つの Pod インスタンスが実行中であることを確認できるレプリケーションコントローラーの例です。

```
id: "webserver-controller"
kind: "ReplicationController"
apiVersion: "v1beta1"
desiredState:
  replicas: 2
  replicaSelector:
    selectorname: "webserver"
  podTemplate:
    desiredState:
      manifest:
        version: "v1beta1"
        id: "webserver-controller"
        containers:
          - name: "apache-frontend"
            image: "webwithdb"
            ports:
              - containerPort: 80
    labels:
      name: "webserver"
      selectorname: "webserver"
      uses: "db"
labels:
  name: "webserver-controller"
```

```
# kubectl create -f webserver-service.yaml
# kubectl get pods
POD         IP          CONTAINER(S)     IMAGE(S)    HOST
       LABELS                                          STATUS
f28980d...  10.20.48.4  apache-frontend  webwithdb   node1.example.com/
       name=webserver,selectorname=webserver,uses=db      Running
f28a0a8...  10.20.29.9  apache-frontend  webwithdb   node2.example.com/
       name=webserver,selectorname=webserver,uses=db      Running
# kubectl get replicationControllers
CONTROLLER          CONTAINER(S)    IMAGE(S)   SELECTOR
```

```
REPLICAS

webserver-controller apache-frontend webwithdb  selectorname=webserver
2
```

注意: この出力では、Pod 名の後ろが切り捨てられ、長い行は折り返して表示されています。

表示された結果がこれと異なる場合は、以下を参照してください。

- **id: missing**: CONTROLLER 列に "webserver-controller" の代わりに生成された数字や文字のセットが表示された場合、yaml ファイルにはおそらく id 行がありません。
- **apiVersion set wrong**: "unable to recognize "webserver-rc.yaml": no object named "ReplicationController" is registered" というメッセージが表示される場合、apiVersion の値が無効か、または ReplicationController の綴りが正しくない可能性があります。
- **selectorname: missing**: "replicationController "webserver-controller" is invalid: spec.selector: required value 'map[]'" というメッセージが表示される場合、replicaSelector 行の後には selectorname が設定されません。selectorname のインデントが適切に行われていない場合、"unable to get type info from "webserver-rc.yaml": couldn't get version/kind: error converting YAML to JSON: yaml: line 7: did not find expected key." のようなメッセージが表示されます。

4.6. トラブルシューティングの手法

Kubernetes クラスターで生じていることを詳細に調べる必要がある場合に参照できる手法を以下に示します。

4.6.1. etcd データベースのクロールおよび修正

etcd サービスは、Kubernetes がクラスター間で情報を統合するために使用するデータベースを提供します。データベースを直接表示して問題を修正する方法があります (または、修正ができない場合はデーターベースを削除します)。

etcd データベースからのデータの表示: kubectl get コマンドを使用して etcd データベースから必要な情報のほとんどを照会できます。ただし、このデータベースが設定が予想したものと異なる場合は、etcdctl コマンドを使用して etcd データベースを直接照会できます。

etcdctl コマンドを ls オプションと共に使用すると、データベースのディレクトリー構造を一覧表示できます。値を取得するには、get オプションを使用します。たとえば、データベースの root を表示するには、以下を入力します。

```
# etcdctl ls /
/coreos.com
/registry
```

etcd データベースの Kubernetes イベントを一覧表示するには、以下を入力します。

```
# etcdctl ls /registry/events/default/

/registry/events/default/679ec44e-e882-11e4-926b-52540...
/registry/events/default/e1f4b268-e87d-11e4-926b-52540...
/registry/events/default/node2.example.com.13d8e2c0b644c20b
/registry/events/default/679ec44e-e882-11e4-926b-52540...
/registry/events/default/5271ec4f-edc1-11e4-8ee6-52540...
/registry/events/default/node1.example.com.13d8e29653ad9bbc
```

データが特定のイベントに関連付けられていることを確認するには、**get** オプションを使用します (**node1** 引数を /registry/events/default/ ディレクトリーのエントリーに置き換えます)。

```
# etcdctl get \
  /registry/events/default/node1.example.com.13d8e29653ad9bbc | \
  python -mjson.tool
{
    "apiVersion": "v1beta1",
    "creationTimestamp": "2015-04-27T09:40:16-04:00",
    "host": "node1.example.com",
    "id": "node1.example.com.13d8e29653ad9bbc",
    "involvedObject": {
        "kind": "Minion",
        "name": "node1.example.com",
        "namespace": "default",
        "uid": "node1.example.com"
    },
    "kind": "Event",
    "message": "Starting kubelet.",
    "namespace": "default",
    "reason": "starting",
    "source": "kubelet",
    "timestamp": "2015-04-27T09:40:16-04:00",
    "uid": "f020d047-ece2-11e4-8ee6-5254001aa4ee"
}
```

上記の出力は python の json.tool フォーマットのモジュールにパイプされ、読みやすくなります。node1.example.com は Minion として定義され、それに関連付けられているイベントが kublet サービスを起動していることを確認できます。

以下は、その他の **etcdctl** を使用した **ls** の実行例です。これらは、レジストリーにあるトップレベルのエントリー、minions の名前、サービスの名前、および使用されているサブネットを示しています。

```
# etcdctl ls /registry/
/registry/events
/registry/minions
/registry/nodes
/registry/pods
/registry/services
/registry/controllers
# etcdctl ls /registry/minions
/registry/minions/node1.example.com
/registry/minions/node2.example.com
# etcdctl ls /registry/services/endpoints/default/
/registry/services/endpoints/default/db-service
/registry/services/endpoints/default/kubernetes
/registry/services/endpoints/default/kubernetes-ro
/registry/services/endpoints/default/webserver-service
# etcdctl ls /coreos.com/network/subnets/
/coreos.com/network/subnets/10.20.1.0-24
/coreos.com/network/subnets/10.20.24.0-24
/coreos.com/network/subnets/10.20.56.0-24
```

以下は、**etcdctl** を使用した **get** の実行例です。最初の例は、db サービスの詳細を示しています。2 つ目の例は、node1 のホスト IP アドレスおよび他の情報について示しています。最後の例は、flannel によって割り当てられるサブネットワークの情報を示しています。

```
# etcdctl get \
    /registry/services/endpoints/default/db-service/  \
    |  python -mjson.tool
{
    "apiVersion": "v1beta1",
    "creationTimestamp": "2015-04-21T18:29:43-04:00",
    "endpoints": [
        "10.20.1.3:3306",
        "10.20.24.3:3306"
    ],
    "id": "db-service",
    "kind": "Endpoints",
    "namespace": "default",
    "resourceVersion": 312,
    "selfLink": "/api/v1beta1/endpoints/db-service?namespace=default",
    "uid": "e87f9435-e875-11e4-926b-5254001aa4ee"
}
# etcdctl get /registry/minions/node1.example.com | python -mjson.tool

{
    "apiVersion": "v1beta1",
    "creationTimestamp": "2015-04-21T16:59:22-04:00",
    "hostIP": "192.168.122.116",
    "id": "node1.example.com",
    "kind": "Minion",
    "resources": {
        "capacity": {
            "cpu": "1",
            "memory": 3221225472
        }
    },
    "status": {},
    "uid": "4926bc0c-e869-11e4-926b-5254001aa4ee"
}
# etcdctl get /coreos.com/network/subnets/10.20.24.0-24 | python -
mjson.tool

{
    "BackendData": {
        "VtepMAC": "86:b1:2e:96:5f:35"
    },
    "BackendType": "vxlan",
    "PublicIP": "192.168.122.116"
}
```

注意: etcdctl が選択できない場合は、**curl** コマンドを代わりに使用できます。たとえば、**curl** でデータベースの root を表示するには、**etcdctl ls /** コマンドの代わりに **curl -L http://master:4001/v2/keys/ | python -mjson.tool** を使用します。この curl コマンドの形式を使用して、ディレクトリーとキー値の両方を表示します。ノードをマスターの etcd サービスに接続できないと思われる場合は、以下の **curl** コマンドを使用して、ノードからその接続をテストできます。

```
# curl -s -L http://master.example.com:4001/version
```

etcd データベースの修正: 情報が同期されない場合、etcd データベースで問題を修正することができます。キーの内容を変更するためのコマンドには、**etcdctl update** および **etcdctl set** コマンドがあります。ただし、注意して使用しないと、これらの値を変更した時に、問題を修正せずに別の問題を発生させることになります。

ただし、etcd データベースが完全に利用できない場合は、一度削除してから再度起動できます。**etcd** デーモンを **-f** オプションと共に実行することにより、これを実行できます。

警告: etcd データベースを削除する前に、**kubectl delete** コマンドを使用して、問題のあるサービス、Pod、レプリケーションコントローラーまたは minions を削除します。データベースを削除する必要が依然としてあると思われる場合は、すべてを最初から作成し直す必要があることに注意してください。

etcd データベースを削除するには、以下を入力します。

```
# etcd -f
```

4.6.2. Kubernetes コンポーネントの削除

Kubernetes のコンポーネントを停止し、削除するには注意が必要です。Kubernetes は物事を特定の状態に維持するように設計されているため、コンテナーや Pod を削除するだけで、別のコンポーネントが起動することがよくあります。そのため、Kubernetes 環境で一部またはすべてのコンポーネントを削除する必要がある場合は、「Kubernetes クラスターの作成」の「Kubernetes の削除」サクションにある指示に従うことをお勧めします。

コンポーネントを削除する順番を間違えると、以下のような問題が発生します。

- **Pod を削除したのに再起動した**: レプリケーションコントローラーを最初に停止しないと Pod が再起動します。レプリケーションコントローラーを停止 (**docker stop replicationControllers webserver-controller**) してから Pod を停止します。
- **コンテナーを停止して削除したのに再起動した**: Kubernetes クラスターを使用する場合、**docker stop** を実行してコンテナーを直接停止することはできません。レプリケーションコントローラーは、新しいコントローラーを起動して、停止したコンテナーを再起動します。

4.6.3. 「WAITING」状態で停止した Pod

Pod は一定期間「waiting (待機状態)」状態になることがあります。以下が考えられる原因です。

- **Docker イメージの取得に時間がかかる**: これを確認するには、Pod が割り当てられている minion に ssh を直接実行し、以下を実行します。

  ```
  # journalctl -f -u docker
  ```

  これにより、イメージを取得する docker のログが表示されます。dockerhub イメージを取得する要求は断続的に失敗する可能性がありますが、kubelets は継続的に再試行します。

- **Pod が割り当てられていない**: Pod が割り当てられていない場合は、kubectl get minions を実行して、ノードがマスターで利用できることを確認します。ノードが停止しているか、または応答しない可能性があります。割り当てられていない Pod は、クラスターが提供できるよりも多くのレプリケーション数を設定することによって発生する可能性もあります。
- **コンテナー Pod が起動直後に終了する**: 作成した Dockerfile が、サービスを起動するために適切に書き込まれていないか、または docker CMD 操作に失敗しているため、起動直後に POD

が終了する場合があります。**docker run** コマンドでコンテナーイメージをテストし、コンテナー自体が壊れていないことを確認します。

- **コンテナーからの出力の確認**: コンテナーからのメッセージの出力は、kubectl log コマンドで表示できます。これは、コンテナー内で実行されているアプリケーションに関する問題をデバッグするのに役立ちます。以下は、利用可能な Pod を一覧表示し、必要な Pod のログメッセージを表示する方法です。

```
# kubectl get pods
POD                                    IP         CONTAINER(S)
IMAGE(S)            HOST                   LABELS
STATUS
e1f4b268-e87d-11e4-926b-5254001aa4ee  10.20.24.3 db
dbforweb node1.example.com/   name=db,selectorname=db
Running
# kubectl log e1f4b268-e87d-11e4-926b-5254001aa4ee
2015-04-28T16:09:36.953130209Z 150428 12:09:36 mysqld_safe Logging
to '/var/log/mariadb/mariadb.log'.
2015-04-28T16:09:37.137064742Z 150428 12:09:37 mysqld_safe Starting
mysqld daemon with databases from /var/lib/mysql
```

- **docker からのコンテナー出力の確認**: 一部のエラーは kubelet にまでは影響しません。終了したコンテナーについて docker ログを直接確認し、どんな問題が発生したかを確認することができます。以下はその方法です。

  - コンテナーの実行に問題のあるノードにログインします。
  - このコマンドを実行して終了した実行を参照します。

```
# docker ps -a
61960bda2927  rhel7/rhel-tools:latest "/usr/bin/bash" 47 hours
ago
      Exited (0) 43 hours ago      myrhel-tools4
```

  - docker logs を使用してコンテナーのすべての出力を確認します。

```
# docker logs 61960bda2927
```

コンテナーセッションから出力全体を確認できるようにする必要があります。たとえば、コンテナーにシェルを開いた場合、docker logs を実行する際にシェルから実行したすべてのコマンドを確認できます。

第5章 YAML の概要

5.1. 概要

YAML (YAML Ain't Markup Language) は、人間にとって読みやすいデータ直列化の標準であり、範囲において JSON (Javascript Object Notation) に類似しています。JSON とは異なる点として、マッピングおよび箇条書きリストを表す少数の特殊文字や 2 つの基本的な構造タイプが使用され、インデントがサブストラクチャーを表すために使用されます。

5.2. 基本情報

YAML 形式は、3 つのハイフンからなる行で区切られる **HEAD** と **BODY** の 2 つのトップレベルの部分で構成される行指向型の形式です。

```
HEAD
---
BODY
```

HEAD は設定情報を保持し、BODY はデータを保持します。本書では設定については論じませんが、ここで取り上げるすべての例はデータ部分のみを示します。このため、“---” はここではオプションになります。

最も基本的なデータ要素は以下のいずれかになります。

1. 数値
2. Unicode 文字列
3. true または false で表されるブール値
4. キー/値のペアのコンテキストでは、見つからない値は nil と解析されます。

コメントは **“#”** (ハッシュ、U+23) で開始され、行末まで続きます。

インデントは行の開始位置にある空白です。**TAB** (U+09) 文字の使用を避け、代わりに一連の **SPACE** (U+20) 文字を使用することを強くお勧めします。

5.3. リスト

リストは一連の行であり、各行は同じ量のインデントとその後のハイフンで始まり、リスト要素がこれに続きます。リストにはブランク行を使用できません。たとえば、以下は 3 つの要素から構成されるリストで、3 つ目の要素にはコメントがあります。

```
- top shelf
- middle age
- bottom dweller   # stability is important
```

注意: 3 つ目の要素は文字列 “bottom dweller” であり、この "dweller” とコメント間に空白は含まれません。

警告: 通常、リストは直接ネストできません。マッピングが介在することがあります (下記に説明)。以下の例では、リストの 2 つ目の要素は、インデント (2 つの **SPACE** 文字) のため、サブリストをホストしているように表示されます。

```
- top
- middle
  - highish middle
  - lowish middle
- bottom
```

しかし、実際には 2 つ目の要素は単一文字列として解析されます。入力は以下のようになります。

```
- top
- middle - highish middle - lowish middle
- bottom
```

改行およびインデントは単一スペースに正規化されます。

5.4. マッピング

マッピング (連想配列またはハッシュテーブルとも呼ばれる) を作成するには、**":"** (コロン、**U+3A**) と、その後に 1 つ以上の **SPACE** 文字をキーと値の間で使用します。

```
square:   4
triangle: 3
pentagon: 5
```

マッピング内のすべてのキーは一意である必要があります。たとえば、以下はキー「square」が繰り返されており、**triangle** のコロンの後にはスペースがないという 2 つの理由により無効な YAML になります。

```
square:  4
triangle:3       # invalid key/value separation
square:  5       # repeated key
```

マッピングは、インデントを増やし、次の行でサブマッピングを開始することにより直接ネストできます。次の例では、キー **square** の値自体がマッピング (キー **sides** および **perimeter**) であり、キー **triangle** の値についても同様です。キー **pentagon** の値は数値 5 です。

```
square:
  sides:     4
  perimeter: sides * side-length
triangle:
  sides:     3
  perimeter: see square
pentagon:    5
```

以下の例は、3 つのキー/値のペアのマッピングを示しています。1 つ目および 3 つ目の値は **nil** であり、2 つ目は「highish middle」および「lowish middle」の 2 つの要素のリストです。

```
top:
middle:
  - highish middle
  - lowish middle
bottom:
```

5.5. 引用符

5.5. 引用符

二重引用符 (“double-quotes”) は、文字列以外のデータを文字列として解釈されるように強制実行したり、空白を保持したり、コロンの意味を抑制したりするのに役立ちます。二重引用符を文字列に組み込むには、`**“\”** (バックスラッシュ、**U+5C**) でエスケープします。以下の例では、すべてのキーおよび値が文字列になります。2 番目のキーにはコロンが含まれます。2 番目の値には、表示されるテキストの前後に 2 つのスペースが含まれます。

```
"true" : "1"
"key the second (which has a \":\" in it)" : "  second  value  "
```

キーを二重引用符で囲む際に読みやすくするには、コロンの前に空白を追加することが奨励されます。

5.6. ブロックコンテンツ

通常は、2 種類のブロックコンテンツがマッピング要素の値の位置にあります。ブロックが **“|”** (パイプ、**U+7C**) で開始する場合、そのブロックの改行は保持されます。**“>”** (より大、**U+3E**) で開始される場合、続く改行は単一スペースにまとめられます。以下の例では、キーの **good-bye** および **anyway** の値であるこれら 2 つの種類のブロックコンテンツを示しています。

```
hello: world

good-bye: |
    first line

    third
    fourth and last


anyway: >
    nothing is guaranteed
    in life
lastly:
```

改行を示す \n (バックスラッシュ-n) を使用すると、キーの **good-bye** および **anyway** の値はそれぞれ以下のようになります。

```
first line\n\nthird\nfourth and last\n

nothing is guaranteed in life\n
```

改行は **good-bye** の値で保持されていますが、**anyway** の値では単一スペースにまとめられていることに注意してください。また、“fourth and last” と “anyway” の間には 2 つのブランク行があり、“in life” と “lastly” 間にはブランク行がなく、それぞれの値は単一の改行で終了しています。

5.7. 簡易表示

リストおよびマッピングを簡易表示する別の方法は、開始文字で始め、終了文字で終了し、要素を **“,”** (コンマ、**U+2C**) で区切る方法です。

リストについては、開始および終了文字はそれぞれ **“[”** (左角かっこ、**U+5B**) および **“]”** (右角括弧、**U+5D**) になります。以下の例では、マッピングで使用される値は同一です。

```
one:
  - echo
  - hello, world!
two: [ echo, "hello, world!" ]
```

注意: 2 つ目の値の 2 つ目のリスト要素周辺の二重引用符については、それらはコンマが要素区切りとして間違って解釈されないように機能します。(二重引用符を削除すると、リストには "echo"、"hello" "world!" の 3 つの要素が含まれてしまいます)。

マッピングについては、開始および終了文字はそれぞれ **“{”** (左中かっこ、**U+7B**) および **“}”** (右中かっこ、**U+7D**) になります。以下の例では、1 つ目および 2 つ目の値は同一になります。

```
one:
  roses: red
  violets: blue

two: { roses: red, violets: blue }
```

5.8. 追加情報

YAML には、ディレクティブ、複雑なマッピングキー、フロースタイル、リファレンス、エイリアス、およびタグなどの本書で扱っていない他の多くの特徴があります。詳細は、正式な YAML サイト、とくに最新 (本書の作成時点では バージョン 1.2) の仕様を参照してください。

第6章 KUBERNETES におけるストレージのプロビジョニング

6.1. 概要

このセクションでは、Kubernetes でのストレージのプロビジョニング方法について説明します。

本書の演習を開始する前に、Kubernetes をセットアップしておく必要があります。

Kubernetes がセットアップされていない場合は、「Kubernetes によるコンテナーのオーケストレーション」の指示に従ってください。

6.2. KUBERNETES 永続ボリューム

このセクションでは、Kubernetes の永続ボリュームについて概観します。以下の例では、**nginx** を使用して永続ボリュームからコンテンツを提供する方法について説明します。

このセクションでは、読者が Kubernetes を基本的に理解されており、Kubernetes クラスターを稼働しておられることを想定しています。

Kubernetes の永続ボリューム (PV) は、インフラストラクチャーの基礎となるストレージ容量の実際の構成要素を表しています。Kubernetes を使用してマウントを実行する前に、まずマウントするストレージを作成する必要があります。クラスター管理者は、Kubernetes でマウントできるよう GCE ディスクを作成し、NFS 共有をエクスポートする必要があります。

永続ボリュームは、GCE 永続ディスク、NFS 共有、および AWS ElasticBlockStore ボリュームなどの「ネットワークボリューム」用に設計されています。HostPath が開発およびテストを容易にするために組み込まれています。以下の例では、ローカルの HostPath を作成します。

重要! HostPath を機能させるには、単一ノードクラスターを実行する必要があります。Kubernetes は現時点ではホスト上のローカルストレージをサポートしません。Pod は HostPath のある正しいノードに常に必ず配置される訳ではありません。

```
// this will be nginx's webroot
$ mkdir /tmp/data01
$ echo 'I love Kubernetes storage!' > /tmp/data01/index.html
```

物理ボリュームを API サーバーに提示して作成します。

```
$ kubectl create -f examples/persistent-volumes/volumes/local-01.yaml
$ kubectl get pv

NAME        LABELS        CAPACITY            ACCESSMODES
STATUS              CLAIM
pv0001      map[]         10737418240         RWO
Available
```

6.2.1. ストレージの要求

Kubernetes のユーザーは、Pod 用に永続ストレージを要求します。基礎となるプロビジョニングの性質については、ユーザーが理解している必要はありません。ただし、ユーザーはストレージの要求を使用でき、ストレージを使用する数多くの Pod とは別にそのストレージのライフサイクルを管理できることを知っておく必要があります。

要求は、それらを使用する Pod と同じ名前空間に作成される必要があります。

```
$ kubectl create -f examples/persistent-volumes/claims/claim-01.yaml
$ kubectl get pvc

NAME                LABELS              STATUS              VOLUME
myclaim-1           map[]
```

バックグラウンドプロセスは、この要求をボリュームに一致させることを試行します。要求の状態は最終的には以下のようになります。

```
$ kubectl get pvc

NAME        LABELS    STATUS    VOLUME
myclaim-1   map[]     Bound     f5c3a89a-e50a-11e4-972f-80e6500a981e


$ kubectl get pv

NAME                LABELS              CAPACITY
ACCESSMODES         STATUS    CLAIM
pv0001              map[]               10737418240         RWO
Bound     myclaim-1 / 6bef4c40-e50b-11e4-972f-80e6500a981e
```

6.2.2. 要求をボリュームとして使用する

要求は Pod のボリュームとして使用されます。Kubernetes は要求を使用してそれがバインドされた PV を検索します。次に、PV は Pod に公開されます。

```
$ kubectl create -f examples/persistent-volumes/simpletest/pod.yaml

$ kubectl get pods

POD       IP           CONTAINER(S)   IMAGE(S)   HOST
LABELS    STATUS    CREATED
mypod     172.17.0.2   myfrontend     nginx      127.0.0.1/127.0.0.1
<none>    Running   12 minutes>


$ kubectl create -f examples/persistent-volumes/simpletest/service.json
$ kubectl get services

NAME              LABELS                                    SELECTOR
IP           PORT(S)
frontendservice   <none>
name=frontendhttp   10.0.0.241   3000/TCP
kubernetes        component=apiserver,provider=kubernetes   <none>
10.0.0.2     443/TCP
kubernetes-ro     component=apiserver,provider=kubernetes   <none>
10.0.0.1     80/TCP
```

6.2.3. 次のステップ

nginx が提供しているコンテンツを表示するためにサービスエンドポイントを照会します。"forbidden" エラーが出た場合は、SELinux (# setenforce 0) を無効にします。

```
# curl 10.0.0.241:3000
I love Kubernetes storage!
```

6.3. ボリューム

Kubernetes は各種のストレージ機能を「ボリューム」として抽象化します。

ボリュームは、Pod 定義の **volumes** セクションで定義されます。ボリュームにあるデータのソースは (1) リモート NFS 共有、(2) iSCSI ターゲット、(3) 空のディレクトリー、または (4) ホスト上のローカルディレクトリーのいずれかになります。

Pod 定義の **volumes** セクションでは複数のボリュームを定義することができます。各ボリュームは、マウント手順の実行時に Pod 内の固有 ID として使用される (Pod のコンテキスト内の) 固有名を持つ必要があります。

これらのボリュームは、いったん定義されると Pod 定義の **containers** セクションに定義されるコンテナーにマウントできます。各コンテナーでは複数のボリュームをマウントできます。一方、単一ボリュームを複数のコンテナーにマウントすることもできます。コンテナー定義の **volumeMounts** セクションでは、ボリュームをマウントする必要のある場所を指定します。

6.3.1. 例

```
apiVersion: v1beta3
kind: Pod
metadata:
  name: nfs-web
spec:
  volumes:
    # List of volumes to use, i.e. *what* to mount
    - name: myvolume
      < volume details, see below >
    - name: mysecondvolume
      < volume details, see below >

  containers:
    - name: mycontainer
      volumeMounts:
        # List of mount directories, i.e. *where* to mount
        # We want to mount 'myvolume' into /usr/share/nginx/html
        - name: myvolume
          mountPath: /usr/share/nginx/html/
        # We want to mount 'mysecondvolume' into /var/log
        - name: mysecondvolume
          mountPath: /var/log/
```

6.4. KUBERNETES および SELINUX パーミッション

Kubernetes が適切に機能するには、ホストとコンテナー間で共有されるディレクトリーへのアクセスがなければなりません。SELinux はデフォルトでは、Kubernetes がその共有ディレクトリーにアクセスすることをブロックします。通常このブロックは得策と言えます。脆弱なコンテナーがホストにアクセスしてダメージを加えることは誰もが阻止したいことであるためです。この状況では、SELinux の共有の阻止に向けた介入なしに、ディレクトリーをホストと Pod 間で共有されるようにする必要があります。

以下は一例になります。ディレクトリー **/srv/my-data** を Atomic Host と Pod で共有する必要がある場合、**/srv/my-data** を SELinux ラベルの **svirt_sandbox_file_t** で、明示的に再度ラベル付けする必要があります。(ホストにある) このディレクトリーにこのラベルが付けられることにより、SELinux はコンテナーに対し、ディレクトリーの読み取りおよびディレクトリーへの書き込みを許可します。以下は、**svirt_sandbox_file_t** ラベルを **/srv/my-data** ディレクトリーに割り当てるコマンドです。

```
$ chcon -R -t svirt_sandbox_file_t /srv/my-data
```

以下の例は、この手順を実行するためのステップを示しています。

ステップ 1

ホストから **/srv/my-data** を使用するコンテナーを HTML ルートとして定義します。

```
{
 "apiVersion": "v1beta3",
 "kind": "Pod",
 "metadata": {
   "name": "host-test"
 },
 "spec": {
   "containers": [
     {
       "name": "host-test",
       "image": "nginx",
       "privileged": false,
       "volumeMounts": [
         {
           "name": "srv",
           "mountPath": "/usr/share/nginx/html",
           "readOnly": false
         }
       ]
     }
   ],
   "volumes": [
     {
       "name": "srv",
       "hostPath": {
         "path": "/srv/my-data"
       }
     }
   ]
 }
}
```

ステップ 2

コンテナーホストで以下のコマンドを実行し、SELinux が nginx コンテナーの**/srv/my-data** への読み取りアクセスを拒否していることを確認します。

```
$ mkdir /srv/my-data
$ echo "Hello world" > /srv/my-data/index.html
$ curl <IP address of the container>
```

以下の出力が表示されます。

```
<html>
<head><title>403 Forbidden</title></head>
...
```

ステップ 3

ラベル **svirt_sandbox_file_t** をディレクトリー **/srv/my-data** に適用します。

```
$ chcon -R -t svirt_sandbox_file_t /srv/my-data
```

ステップ 4

curl を使用してコンテナーにアクセスし、ラベルが有効になったことを確認します。

```
$ curl <IP address of the container>
Hello world
```

curl コマンドが "Hello world" を返す場合、SELinux ラベルは適切に適用されていることになります。

詳細は、この点についての情報を追跡している BZ#1222060[を参照してください。

6.5. NFS

以下のシナリオをテストするには、NFS 共有が事前に準備されている必要があります。この例では、NFS 共有を Pod にマウントします。

以下の例では、NFS 共有を **/usr/share/nginx/html/** にマウントし、**nginx** webserver を実行します。

ステップ 1

nfs-web.yaml という名前のファイルを作成します。

```
apiVersion: v1beta3
kind: Pod
metadata:
  name: nfs-web
spec:
  volumes:
    - name: www
      nfs:
        # Use real NFS server address here.
        server: 192.168.100.1
        # Use real NFS server export directory.
        path: "/www"
        readOnly: true
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: tcp
```

```
      volumeMounts:
          # 'name' must match the volume name below.
          - name: www
            # Where to mount the volume.
            mountPath: "/usr/share/nginx/html/"
```

ステップ 2

Pod の起動:

```
$ kubectl create -f nfs-web.yaml
```

Kubernetes は **`192.168.100.1:/www`** を nginx コンテナー内の **`/usr/share/nginx/html/`** ` にマウントし、これを実行します。

ステップ 3

webserver が NFS 共有からデータを受信することを確認します。

```
$ curl 172.17.0.6
Hello from NFS
```

トラブルシューティング

403 Forbidden error: webserver から "403 Forbidden" の応答を受信する場合、以下のコマンドを実行し、SELinux が Docker コンテナーに対して NFS 経由のデータの読み取りを許可していることを確認します。

```
$ setsebool -P virt_use_nfs 1
```

6.6. ISCSI

ステップ 1 iSCSI ターゲットが適切に設定されていることを確認します。すべての Kubernetes ノードに iSCSI ターゲットから LUN を割り当てる十分な特権があることを確認します。

ステップ 2 以下の Pod 定義を含む **`iscsi-web.yaml`** という名前のファイルを作成します。

```
apiVersion: v1beta3
kind: Pod
metadata:
  name: iscsi-web
spec:
  volumes:
    - name: www
      iscsi:
        # Address of the iSCSI target portal
        targetPortal: "192.168.100.98:3260"
        # IQN of the portal
        iqn: "iqn.2003-01.org.linux-iscsi.iscsi.x8664:sn.63b56adc495d"
        # LUN we want to mount
        lun: 0
        # Filesystem on the LUN
        fsType: ext4
        readOnly: false
  containers:
```

```
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: tcp
      volumeMounts:
          # 'name' must match the volume name below.
          - name: www
            # Where to mount he volume.
            mountPath: "/usr/share/nginx/html/"
```

ステップ 3

Pod の作成:

```
$ kubectl create -f iscsi-web.yaml
```

ステップ 4

Kubernetes は iSCSI ターゲットにログインし、LUN 0 を割り当て (通常は **/dev/sdXYZ**)、指定のファイルシステム (本書の例では ext4) を nginx コンテナー内の **/usr/share/nginx/html/** にマウントして、これを実行します。

ステップ 5

web サーバーが iSCSI ボリュームのデータを使用していることを確認します。

```
$ curl 172.17.0.6
Hello from iSCSI
```

6.7. GOOGLE COMPUTE ENGINE

Google Compute Engine 永続ディスク (GCE PD)

クラスターを Google Compute Engine で実行している場合、永続ディスクを永続ストレージソースとして使用できます。以下の例では、GCE PD から html コンテンツを提供する Pod を作成します。

ステップ 1

GCE SDK がセットアップされている場合、以下のコマンドを使用して永続ディスクを作成します。

```
$ gcloud compute disks create --size=250GB {Persistent Disk Name}
```

または、GCE web インターフェースでディスクを作成することもできます。GCE SDK をセットアップする必要がある場合は、こちら の指示に従ってください。

ステップ 2

gce-pd-web.yaml という名前のファイルを作成します。

```
apiVersion: v1beta3
kind: Pod
metadata:
```

```
  name: gce-web
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: tcp
      volumeMounts:
        - name: html-pd
          mountPath: "/usr/share/nginx/html"
  volumes:
    - name: html-pd
      gcePersistentDisk:
        # Add the name of your persistent disk below
        pdName: {Persistent Disk Name}
        fsType: ext4
```

ステップ 3

Pod の作成:

```
$ kubectl create -f gce-pd-web.yaml
```

Kubernetes は Pod を作成し、ディスクを割り当てますが、これをフォーマットしたり、マウントしたりすることはできません。これはバグによる問題ですが、Kubernetes の今後のバージョンで修正される予定です。この問題を回避するために、次のステップに進んでください。

ステップ 4

永続ディスクをフォーマットし、マウントします。

ステップ 5

ディスクは仮想マシンに割り当てられ、デバイスは **scsi-0Google_PersistentDisk_{Persistent Disk Name}** という名前で **/dev/disk/by-id/`** に表示されます。このディスクがすでにフォーマットされており、データが含まれている場合は、次のステップに進みます。そうでない場合は root で以下のコマンドを実行し、これをフォーマットします。

```
$ mkfs.ext4 /dev/disk/by-id/scsi-0Google_PersistentDisk_{Persistent
Disk Name}
```

ステップ 6

ディスクがフォーマットされている場合、これを Kubernetes が予想する場所にマウントします。以下のコマンドを root として実行します。

```
# mkdir -p /var/lib/kubelet/plugins/kubernetes.io/gce-
pd/mounts/{Persistent Disk Name} && mount /dev/disk/by-id/scsi-
0Google_PersistentDisk_{Persistent Disk Name}
/var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk
Name}
```

注意: mkdir コマンドおよび mount コマンドは上記のように連続して実行される必要があります。Kubernetes の削除により、何もマウントされていないことが確認されるとディレクトリーが削除

されるためです。

ステップ 7

ディスクがマウントされており、正しい SELinux コンテキストが付与されているはずです。root で以下を実行します。

```
$ sudo chcon -R -t svirt_sandbox_file_t
/var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk
Name}
```

ステップ 8

webserver が提供するデータを作成します。

```
$ echo "Hello world" >  /var/lib/kubelet/plugins/kubernetes.io/gce-
pd/mounts/{Persistent Disk Name}/index.html
```

ステップ 9

Pod から HTML を取得できるはずです。

```
$ curl {IP address of the container}
Hello World!
```

第7章 DOCKER フォーマットのコンテナーイメージの使用方法

7.1. 概要

昨今 Docker はアプリケーションのコンテナー化における重要なプロジェクトの 1 つとなっています。本書では、Red Hat Enterprise Linux 7 および RHEL Atomic で Docker を使用するための実践的なアプローチを紹介します。ここでは、Docker レジストリーのセットアップ、Docker イメージの取得と使用、および Docker コンテナーの使用について説明します。

7.2. 背景

Docker プロジェクトは、軽量コンテナーでアプリケーションをパッケージ化する方法を提供します。アプリケーションを Docker コンテナーで実行すると、以下のような利点があります。

- **仮想マシンよりもサイズが小さい**: Docker イメージにはアプリケーションの実行に必要なコンテナーのみが含まれるため、(オペレーティングシステム全体を含む) 仮想マシンの場合よりも Docker を使用して保存および共有を実行する方がはるかに効率的です。
- **パフォーマンスの向上**: 同様にコンテナーの場合は全く別個のオペレーティングシステムを実行する必要がないため、通常は新規の仮想マシン全体のオーバーヘッドを伴うアプリケーションよりも高速に実行されます。
- **安全性**: Docker コンテナーには通常、独自のネットワークインターフェース、ファイルシステムおよびメモリーがあるため、そのコンテナーで実行されるアプリケーションはホストコンピューターの他のアクティビティーから分離し、セキュリティーを保護することができます。
- **柔軟性**: コンテナー内のアプリケーションに組み込まれているアプリケーションのランタイム要件により、Docker コンテナーは複数の環境で実行できます。

現時点で、Docker コンテナーは Red Hat Enterprise Linux 7 (RHEL 7) および Red Hat Enterprise Linux Atomic (RHEL 7 ベース) システムで実行できます。RHEL Atomic に馴染みがない場合は、『Red Hat Enterprise Linux Atomic Host スタートガイド』またはアップストリームの Project Atomic サイトを参照してください。Project Atomic では、とくに OpenStack、VirtualBox、Linux KVM その他のいくつかの異なる環境で Docker コンテナーを実行することを目的として作成された RPM ベースの Linux ディストリビューション (RHEL、Fedora および CentOS) の小型版を生成しています。

本書は、RHEL 7 および RHEL Atomic の初期リリースの Docker の使用を開始するのに役立ちます。Docker を試用する実践的な方法のほかに、以下を実行する方法について説明します。

- Red Hat カスタマーポータルから RHEL ベースの Docker イメージへのアクセス
- RHEL で有効なソフトウェアのコンテナーへの組み込み

本書の今後のリリースは、以下に役立つ情報が提供されます。

- RHEL のセキュリティー機能によるコンテナーの安全なデプロイメント
- Docker イメージの構築方法を標準化するツールの検索
- セキュリティープロトコルに準拠する方法でコンテナーを構築するためのヒント

Docker の仕組みの詳細をお知りになりたい場合は、以下を参照してください。

- **リリースノート**: RHEL 7 の Docker 機能の概要については、RHEL 7 リリースノートの Docker フォーマットの Linux コンテナー のセクションを参照してください。

- **Docker プロジェクトサイト**: Docker サイト の What is Docker? ページおよび Getting Started ページから Docker の詳細を学ぶことができます。さらに、Docker ドキュメンテーション ページを参照することもできます。
- **Docker README**: docker パッケージのインストール後は、/usr/share/doc/docker-1* ディレクトリーの README.md ファイルを参照してください。
- **Docker の man ページ**: Docker をインストールした状態で **`man docker`** と入力すると、docker コマンドについて確認できます。次に各 Docker オプションについてそれぞれの man ページを参照します (たとえば **`man docker-image`** と入力し、**`docker image`** オプションについて確認します)。

注意: 現時点で、docker コマンドを RHEL 7 および RHEL Atomic で実行するには root 権限が必要です。この手順ではコマンドプロンプトにハッシュ記号 (#) がある場合は root 権限が必要です。root ユーザーアカウントに直接ログインすることを選択しない場合は、sudo を設定することができます。

7.3. RHEL 7 における DOCKER の取得

Docker を開発できる環境を取得するには、コンテナーホストだけではなく開発システムとして機能するように Red Hat Enterprise Linux 7 システムをインストールすることができます。Docker パッケージ自体は RHEL Extras リポジトリーに保存されます (Red Hat Enterprise Linux Extras チャンネルのサポートポリシーおよびライフサイクル情報については Red Hat Enterprise Linux Extras Life Cycle の記事を参照してください)。

RHEL 7 サブスクリプションモデルを使用して Docker イメージまたはコンテナーを作成する必要がある場合は、それらを構築するために使用するホストコンピューターを適切に登録し、これにエンタイトルメントを付与する必要があります。コンテナー内で **yum install** を使用してパッケージを追加する場合、コンテナーは RHEL 7 ホストから利用できるエンタイトルメントに自動的にアクセスできるため、ホストで有効にされたすべてのリポジトリーから RPM パッケージを取得できます。

1. **RHEL Server エディションのインストール**: 開始する準備ができたら、Red Hat Enterprise Linux 7 インストールガイド で説明されているように Red Hat Enterprise Linux システム (Server エディション) をインストールして開始することができます。
2. **RHEL の登録**: RHEL 7 がインストールされたら、サブスクリプション管理ツールを使用してシステムを登録し、Docker パッケージをインストールします。さらに、必要なソフトウェアリポジトリーを有効にします(pool_id を RHEL 7 サブスクリプションのプール ID に置き換えます)。以下のようになります。

   注意: 本書では、docker および docker-registry サービスが同じホストシステムで実行されていることを示しています。実際の環境ではよくあることですが、Docker レジストリーが複数のクライアントで使用されている場合は、docker-registry をインストールするシステムとこれを実行するシステムを別にすることができます。この場合、docker-registry パッケージを docker を実行するシステムにインストールする必要はありません。

   ```
   # subscription-manager register --username=rhnuser --
   password=rhnpasswd
   # subscription-manager list --available  Find pool ID for RHEL
   subscription
   # subscription-manager attach --pool=pool_id
   # subscription-manager repos --enable=rhel-7-server-extras-rpms
   # subscription-manager repos --enable=rhel-7-server-optional-rpms
   ```

   注意: Red Hat Satellite 5 の docker パッケージを取得するのに必要なチャンネル名について

は、Satellite 5 repo to install Docker on Red Hat Enterprise Linux 7 を参照してください。

3. **docker および docker-registry のインストール**: docker パッケージをインストールし、オプションで docker-registry をインストールします。(device-mapper-libs および device-mapper-をインストールしていない場合はここでインストールします。)

```
# yum install docker docker-registry
# yum install device-mapper-libs device-mapper-event-libs
```

4. **docker の起動**:

```
# systemctl start docker.service
```

5. **docker の有効化**:

```
# systemctl enable docker.service
```

6. **docker ステータスの確認**:

```
# systemctl status docker.service
    docker.service - Docker Application Container Engine
       Loaded: loaded (/usr/lib/systemd/system/docker.service;
enabled)
       Active: active (running) since Thu 2014-10-23 11:32:11
EDT; 14s ago
         Docs: http://docs.docker.io
     Main PID: 2068 (docker)
       CGroup: /system.slice/docker.service
               └─2068 /usr/bin/docker -d --selinux-enabled -H
fd://
    ...
```

docker サービスを実行すると、一部の Docker イメージを取得し、**docker** コマンドを使用して RHEL 7 で Docker イメージを使用できるようになります。

7.4. RHEL ATOMIC における DOCKER の取得

RHEL Atomic は、コンテナーを実行するために特別に設計された軽量の Linux オペレーティングシステムのディストリビューションです。これには、Docker サービスと Kubernetes および Etcd サービスを含む、Docker コンテナーのオーケストレーションと管理を行うために使用できるいくつかのサービスが含まれます。

RHEL Atomic はフル機能の Linux システムではなくアプライアンスであるため、(システムに追加するコンテナー以外の) RPM パッケージまたはその他のソフトウェアをインストールするために作成されている訳ではありません。

RHEL Atomic には既存のパッケージをアップデートするメカニズムがありますが、これはユーザーが新規パッケージを追加するためのものではありません。したがって、(開発およびデバッグツールを追加するには) 標準の RHEL 7 サーバーシステムを使用してアプリケーションを開発することを検討してください。標準の RHEL 7 サーバーシステムを使用すれば、RHEL Atomic を使用してコンテナーをさまざまな仮想化環境とクラウド環境にデプロイできます。

つまり RHEL Atomic システムをインストールすれば、コンテナーを実行し、構築し、停止し、開始し、本書で紹介している例に従ってコンテナーを使用したりすることができます。そのためには、以下の手順を使用して RHEL Atomic を取得し、インストールします。

1. **RHEL Atomic の取得**: RHEL Atomic は Red Hat カスタマーポータルから使用できます。RHEL Atomic をライブイメージ (.qcow2 形式) として実行するか、またはインストールメディア (.iso 形式) から RHEL Atomic をインストールするオプションがあります。以下のサイトからこれらの形式 (およびその他の形式) で RHEL Atomic を取得できます。

   RHEL Atomic Host Downloads

   次に『Red Hat Enterprise Linux Atomic Host スタートガイド』に従って Atomic をセットアップし、いくつかの異なる仮想環境で実行します。

2. **RHEL Atomic の登録**: RHEL 7 Atomic がインストールされたら、サブスクリプション管理ツールを使用してシステムを登録します (これにより、**atomic upgrade** を実行して Atomic ソフトウェアをアップグレードできますが、yum コマンドを使用して追加パッケージをインストールすることはできません)。以下は例になります。

   ```
   # subscription-manager register --username=rhnuser --
   password=rhnpasswd --auto-attach
   ```

   重要: 本書で説明しているように、docker コマンドでコンテナーを実行するのに、RHEL Atomic システムを登録したり、サブスクリプションを割り当てたりする必要はありません。ただし、コンテナーで **yum install** コマンドを実行する必要がある場合は、コンテナーは RHEL Atomic ホストから有効なサブスクリプション情報を取得する必要があります。この情報がないと失敗します。ホストが使用している RHEL バージョンでデフォルトで有効にされないリポジトリーを有効にする必要がある場合は、**/etc/yum.repos.d/redhat.repo** ファイルを編集する必要があります。これはコンテナー内で手動で実行でき、使用するレポジトリーに enabled=1 を設定します。さらに、yum repo ファイルを管理するためにコマンドラインツールの **yum-config-manager** を使用することもできます。以下のコマンドを使用してリポジトリーを有効にできます。

```
# yum-config-manager --enable REPOSITORY

You can also use **yum-config-manager** to display Yum global options,
add repositories and others. **yum-config-manager** is documented in
detail
in the Red Hat Enterprise Linux 7 System Administrator's Guide. Since
**redhat.repo** is a big file and editing it manually can be error
prone,
it is recommended to use **yum-config-manager**.
```

1. **Docker の使用の開始**: RHEL Atomic には docker パッケージがすでにインストールされ、有効にされています。そのため、ログインして Atomic システムのサブスクライブを行っている場合は、docker と関連ソフトウェアのステータスは以下のようになります。

   - docker コマンドを実行して、Docker イメージとコンテナーをすぐに使用できます。
   - docker-registry パッケージはインストールされていません。Atomic システムとプライベートのレジストリー間でイメージを移動できるようにするには、docker-registry パッケージを RHEL 7 システムにインストールし (以下で説明)、独自のコンテナーイメージを保存するためにレジストリーにアクセスします。
   - Docker コンテナーのオーケストレーションに使用される kubernetes パッケージは RHEL Atomic にインストールされますが、デフォルトでは有効にされません。RHEL Atomic のコンテナーを Kubernetes でオーケストレーションできるようにするには、いくつかの Kubernetes 関連のサービスを有効にし、起動する必要があります。

7.5. DOCKER レジストリーの使用

Docker レジストリーは、他のユーザーと共有できるイメージとして保存される docker コンテナーを保存し、共有するための場所を提供します。docker パッケージを RHEL および RHEL Atomic で利用可能な状態にすると、イメージを Red Hat カスタマーポータルから取得し、各自のプライベートレジストリーからイメージの移動を実行できます。Red Hat Container イメージの検索ページ を検索して、Red Hat カスタマーポータルから取得できるイメージを確認できます (**docker pull** を使用)。

このセクションでは、ローカルレジストリーの起動方法、Docker イメージをローカルレジストリーにロードする方法、およびそれらのイメージを使用して docker コンテナーを起動する方法について説明します。

7.5.1. Docker のプライベートレジストリーの作成

Docker のプライベートレジストリーを作成する 1 つの方法は、docker-registry サービスを使用する方法です。本書の始めの部分で説明されているように docker-registry パッケージを RHEL 7 (Atomic では利用不可) にインストールしている場合、以下のようにサービスを有効にし、起動することができます。

1. **docker-registry サービスの有効化および起動**: 以下を入力すると、docker-registry サービスを有効にし、起動し、ステータスを確認することができます。

```
# systemctl enable docker-registry
# systemctl start docker-registry
# systemctl status docker-registry
docker-registry.service - Registry server for Docker
   Loaded: loaded (/usr/lib/systemd/system/docker-
registry.service; enabled)
   Active: active (running) since Thu 2014-10-23 13:40:26 EDT; 4s
ago
Main PID: 21031 (gunicorn)
   CGroup: /system.slice/docker-registry.service
           ├─21031 /usr/bin/python /usr/bin/gunicorn --access-
logfile - --debug ...
   ...
```

2. **レジストリー、ファイアウォールの問題**: docker-registry サービスは TCP ポート 5000 で listen するため、そのポートへのアクセスは、ローカルシステム外のクライアントがレジストリーを使用できるように開いた状態にしておく必要があります。これは、docker-registry や docker を同じシステムで実行しているかどうかとは関係がありません。以下を実行して TCP ポート 5000 を開くことができます。

```
# firewall-cmd --zone=public --add-port=5000/tcp
# firewall-cmd --zone=public --add-port=5000/tcp --permanent
# firewall-cmd --zone=public --list-ports
5000/tcp
```

または、iptables ファイアウォールルールを直接使用しているファイアウォールを有効にしている場合は、システムを起動するたびに以下のコマンドを実行する方法があります。

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 5000
-j ACCEPT
```

7.5.2. リモート Docker レジストリーからのイメージの取得

リモートレジストリー (Red Hat 独自の Docker レジストリー) から Docker イメージを取得し、それらをローカルシステムに追加するには、**docker pull** コマンドを使用します。

```
# docker pull <registry>[:<port>]/[<namespace>/]<name>:<tag>
```

<registry> は、TCP **<port>** に docker-registry サービスを提供するホストです (デフォルト: 5000)。さらに、**<namespace>** および **<name>** は、そのレジストリーで **<namespace>** によって制御される特定のイメージを特定します。また、一部のレジストリーも生の **<name>** をサポートします。これらの場合、**<namespace>** はオプションです。ただし、名前空間が組み込まれていると、**<namespace>** が提供する階層の追加のレベルは、同じ **<name>** の複数のイメージを区別するのに役立ちます。以下は例になります。

名前空間	例 **(<namespace>/<name>)**
組織	**redhat/kubernetes**, **google/kubernetes**
ログイン (ユーザー名)	**alice/application**, **bob/application**
ロール	**devel/database**, **test/database**, **prod/database**

2014 年 11 月の時点で、Red Hat がサポートする唯一の Docker レジストリーは registry.access.redhat.com にあります。tarball として保存されている Docker イメージにアクセスできる場合、そのイメージをローカルファイルシステムから Docker レジストリーにロードできます。

docker pull: pull オプションを使用すると、リモートレジストリーからイメージを取得できます。rhel ベースイメージを Red Hat レジストリーから取得するには、**docker pull registry.access.redhat.com/rhel** と入力します。イメージが Red Hat レジストリーのものであることを確認するには、レジストリーのホスト名、スラッシュおよびイメージ名を入力します。以下のコマンドは、**rhel** イメージを Red Hat レジストリーから取得する例を示しています。

```
# docker pull registry.access.redhat.com/rhel
```

イメージはリポジトリー名とタグで特定されます。リポジトリー名 **rhel** は、レジストリー名を付けずに **docker pull** コマンドに渡されると、不明確となり、信用できないリポジトリーのイメージの取得を招く可能性があります。したがって、**latest** などのタグを追加して、名前を **rhel:latest** にします。

上記の **docker pull** コマンドで作成されたイメージを表示するには、**docker images** と入力します。

```
# docker images
REPOSITORY                        TAG         IMAGE ID      CREATED
VIRTUAL SIZE
registry.access.redhat.com/rhel  0-21       e1f5733f050b  4 months ago
```

```
140.2 MB
registry.access.redhat.com/rhel  0          bef54b8f8a2f  4 months ago
139.6 MB
registry.access.redhat.com/rhel  0-23       bef54b8f8a2f  4 months ago
139.6 MB
registry.access.redhat.com/rhel  latest     bef54b8f8a2f  4 months ago
139.6 MB
```

docker load: コンテナーイメージをローカルシステムに tarball として保存する場合、そのイメージの tarball をロードして、ローカルシステムで docker コマンドで実行できます。以下の手順に従ってください。

1. 現在のディレクトリーにある Docker イメージの tarball を使用して、以下のように tarball をローカルシステムにロードします。

```
# docker load -i rhel-server-docker-7.0-23.x86_64.tar.gz
```

2. ローカルホストで実行されているレジストリーに同じイメージを送る場合は、docker-registry サービスのホスト名 (または "localhost") とポート番号 (TCP ポート 5000) のタグをイメージに付けます。**docker push** はこのタグ情報を使用して、イメージを適切なレジストリーに送ります。

```
# docker tag bef54b8f8a2f localhost:5000/myrhel7
docker push localhost:5000/myrhel7
The push refers to a repository [localhost:5000/myrhel7] (len: 1)
Sending image list
Pushing repository localhost:5000/myrhel7 (1 tags)
bef54b8f8a2f: Image successfully pushed
Pushing tag for rev [bef54b8f8a2f] on
{http://localhost:5000/v1/repositories/myrhel7/tags/latest}
...
```

7.5.3. Docker イメージの調査

イメージをローカルレジストリーに追加し、ロードしている場合、docker コマンドの **docker images** を使用してそれらのイメージを表示できます。以下は、ローカルシステムにあるイメージを一覧表示する方法を示しています。

```
# docker images
REPOSITORY   TAG      IMAGE ID      CREATED       VIRTUAL SIZE
redhat/rhel  latest   e1f5733f050b  4 weeks ago   140.2 MB
rhel         latest   e1f5733f050b  4 weeks ago   140.2 MB
redhat/rhel7 0        e1f5733f050b  4 weeks ago   140.2 MB
redhat/rhel7 0-21     e1f5733f050b  4 weeks ago   140.2 MB
redhat/rhel7 latest   e1f5733f050b  4 weeks ago   140.2 MB
rhel7        0        e1f5733f050b  4 weeks ago   140.2 MB
rhel7        0-21     e1f5733f050b  4 weeks ago   140.2 MB
rhel7        latest   e1f5733f050b  4 weeks ago   140.2 MB
```

注意: イメージまたはリポジトリーをアップストリームの Docker.io レジストリー (**docker push**) に追加するデフォルトのオプションは、docker コマンドの Red Hat バージョンでは無効になっています。イメージを特定のレジストリーに追加するには、レジストリー、そのポート番号およびタグを指定して、追加するイメージを指定します。

7.5.4. Docker 環境の調査

ここまでで docker および docker-registry サービスを実行でき、いくつかのコンテナーが利用できる状態になったので、Docker 環境とコンテナーの構成内容を確認することができます。**docker** を **version** および **info** オプションと共に実行すると、Docker 環境の状態を確認できます。

docker version: version オプションを使用すると、インストールされている Docker コンポーネントのバージョンを表示できます。なお、 docker パッケージは新しいバージョンを利用できます (RHEL 7 では **yum update docker** でアップデートできます)。

```
# docker version      Shows components/versions in use. Note that
docker needs updating here.
Client version: 1.2.0
Client API version: 1.14
Go version (client): go1.3.1
Git commit (client): 2a2f26c/1.2.0
OS/Arch (client): linux/amd64
Server version: 1.2.0
Server API version: 1.14
Go version (server): go1.3.1
Git commit (server): 2a2f26c/1.2.0
Last stable version: 1.3.0, please update docker
```

docker info: info オプションを使用すると、ローカルコンテナーやイメージの数などのコンポーネントの場所や、Docker ストレージ領域のサイズや場所の情報を確認できます。

```
# docker info
Containers: 3
Images: 5
Storage Driver: devicemapper
 Pool Name: docker-253:1-16826017-pool
 Pool Blocksize: 64 Kb
 Data file: /var/lib/docker/devicemapper/devicemapper/data
 Metadata file: /var/lib/docker/devicemapper/devicemapper/metadata
 Data Space Used: 1042.4 Mb
 Data Space Total: 102400.0 Mb
 Metadata Space Used: 1.3 Mb
 Metadata Space Total: 2048.0 Mb
Execution Driver: native-0.2
Kernel Version: 3.10.0-123.8.1.el7.x86_64
Operating System: Red Hat Atomic Host 7.0
```

7.5.5. Docker コンテナーの使用

(実行しているかどうかに関わらず) システムに作成された Docker イメージは、複数の方法で管理できます。docker run コマンドを実行すると、コンテナーで実行するコマンドを指定できます。コンテナーを起動したら、停止、開始、再起動を行うことができます。不要なコンテナーは削除できます。

Docker コンテナーの実行

docker run コマンドを実行すると、基本的に Docker イメージからコンテナーを新たに作成できます。コンテナーはイメージのコンテンツと、**docker run** コマンドラインで渡す追加のオプションに基づく複数の機能で構成されます。

docker run コマンドラインで渡すコマンドは、コンテナーの内部をその実行環境として表示しま

す。そのため、デフォルトではホストシステムについてはほとんど確認できません。たとえば、デフォルトでは実行中のアプリケーションで以下を確認できます。

- Docker イメージで提供されるファイルシステム。
- コンテナー内の新規のプロセステーブル (ホストのプロセスは表示されない)。
- 新規のネットワークインターフェース (デフォルトでは、別の docker ネットワークインターフェースが DHCP 経由で各コンテナーにプライベートの IP アドレスを提供する)。

コンテナーで利用できるホストからディレクトリーを作成し、コンテナーからホストにネットワークポートをマップし、コンテナーが使用できるメモリーのサイズを制限し、コンテナーで利用できる CPU 共有を拡大する必要などがある場合、これらを **docker run** コマンドラインで実行することができます。以下は、異なる機能を有効にする docker コマンドラインのいくつかの例になります。

例 #1 (quick コマンドの実行): この docker コマンドは **ip addr show eth0** コマンドを実行して、RHEL イメージから生成されるコンテナー内の eth0 ネットワークのアドレス情報を表示します。これは必要最低限のコンテナーであるため、このデモの RHEL 7 ホストシステムから **/usr/sbin** ディレクトリーをマウントします (マウントは -v オプションで実行します)。これには実行に必要な **ip** コマンドが含まれるためです。コンテナーがコマンドを実行した後に、IP アドレス (**172.17.0.2/16**) および eth0 についての他の情報が表示され、コンテナーは停止し、削除されます (**--rm**)。

```
# docker run -v /usr/sbin:/usr/sbin \
    --rm rhel /usr/sbin/ip addr show eth0
20: eth0:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 4e:90:00:27:a2:5d brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.10/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::4c90:ff:fe27:a25d/64 scope link tentative
       valid_lft forever preferred_lft forever
```

このコンテナーを保存して再度使用する場合は、これに名前を割り当て、その名前を使って後で起動することができます。たとえば、このコンテナーに myipaddr という名前を付けます。

```
# docker run -v /usr/sbin:/usr/sbin \
    --name=myipaddr rhel /usr/sbin/ip addr show eth0
20: eth0:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 4e:90:00:27:a2:5d brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.10/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::4c90:ff:fe27:a25d/64 scope link tentative
       valid_lft forever preferred_lft forever
# docker start -i myipaddr
22: eth0:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 4e:90:00:27:a2:5d brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.10/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::4c90:ff:fe27:a25d/64 scope link tentative
       valid_lft forever preferred_lft forever
```

例 #2 (コンテナー内でシェルを実行): コンテナーを使用して bash シェルを起動すると、コンテナー内を確認でき、コンテンツを変更することができます。ここでは、コンテナーの名前を **mybash** に設定しています。**-i** でインタラクティブなセッションを作成し、**-t** でターミナルセッションを開きます。**-i** を使用しないと、シェルは開いた後に終了します。**-t** を使用しないと、シェルは開いたままになりますが、シェルには何も入力できなくなります。

コマンドを実行すると、シェルプロンプトが表示され、コンテナー内からコマンドを実行できるようになります。

```
# docker run --name=mybash -it rhel /bin/bash
[root@49830c4f9cc4/]#
```

ベースの RHEL イメージ内で利用できるアプリケーションはほとんどありませんが、**yum** コマンドを使用してソフトウェアを追加できます。コンテナー内のシェルを開いたら、以下のコマンドを実行します。

```
[root@49830c4f9cc4/]# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.1 (Maipo)
[root@49830c4f9cc4/]# ps
bash: ps: command not found
[root@49830c4f9cc4/]# yum install -y procps
[root@49830c4f9cc4/]# ps -ef
UID      PID    PPID  C STIME TTY       TIME  CMD
root       1       0  0 15:36 ?     00:00:00  /bin/bash
root      46       1  0 15:43 ?     00:00:00  ps -ef
[root@49830c4f9cc4/]# exit
```

コンテナーが RHEL 7.1 コンテナーになっていることに注意してください。**ps** コマンドは RHEL のベースイメージには含まれていません。ただし、上記のように **yum** を使用してインストールできます。コンテナーを終了するには、**exit** と入力します。

コンテナーを終了すると実行しなくなりますが、コンテナーは新規のソフトウェアパッケージがインストールされた状態で存在したままになります。**docker ps -a** を使用してコンテナーを一覧表示します。

```
# docker ps -a
CONTAINER ID  IMAGE        COMMAND      CREATED        STATUS
PORTS  NAMES
49830c4f9cc4  rhel:latest  "/bin/bash"  2 minutes ago  Exited (0) 25
minutes ago        mybash
...
```

docker start を **-ai** オプションと共に使用してコンテナーを再起動できます。以下は例になります。

```
# docker start -ai mybash
[root@a0aee493a605/]#
```

例 #3 (ログファイルのバインドマウント): コンテナー内のログメッセージをホストシステムで利用できるようにする 1 つの方法は、コンテナー内でホストの /dev/log デバイスをバインドマウントする方法です。この例では、ログメッセージを生成する **log_test** という名前の RHEL コンテナーでアプリケーションを実行 (この場合は logger コマンドを使用) し、ホストからコンテナーにマウントした /dev/log デバイスにメッセージを移動する方法について示しています。**--rm** オプションを使用すると、コンテナーが実行後に削除されます。

```
# docker run --name="log_test" -v /dev/log:/dev/log --rm rhel logger
"Testing logging to the host"
# journalctl -b | grep Testing
Apr 22 16:00:37 node1.example.com logger[102729]: Testing logging to
the host
```

Docker コンテナー外からの調査

1 つ以上の Docker コンテナーがホストで実行されているとします。ホストシステムからコンテナーを使用するには、シェルを開いて以下のコマンドをいくつかを実行できます。

docker ps: ps オプションは、現在実行中のすべてのコンテナーを表示します。

```
# docker ps
CONTAINER ID IMAGE       COMMAND     CREATED         STATUS
PORTS    NAMES
0d8b2ded3af0 rhel:latest "/bin/bash" 10 minutes ago  Up 3 minutes
mybash
```

実行されていないものの、(--rm オプションを使用して) 削除されていないコンテナーがある場合、コンテナーは保存されているために再起動できます。**docker ps -a** コマンドは、実行中または停止しているすべてのコンテナーを表示します。

```
# docker ps -a
CONTAINER ID  IMAGE       COMMAND    CREATED      STATUS
PORTS  NAMES
92b7ed0c039b  rhel:latest /bin/bash  2 days ago   Exited (0) 2 days ago
agitated_hopper
eaa96236afa6  rhel:latest /bin/bash  2 days ago   Exited (0) 2 days ago
prickly_newton
```

システムに存在するコンテナーの起動、停止、および削除についての詳細は、「Docker コンテナーの使用」セクションを参照してください。

docker inspect: 既存のコンテナーのメタデータを調べるには、**docker inspect** コマンドを使用します。コンテナーのすべてのメタデータを表示することも、選択した項目のみを表示することもできます。たとえば、選択したコンテナーのすべてのメタデータを表示するには、以下を入力します。

```
# docker inspect mybash
[{
    "Args": [],
        ...
        "Hostname": "a0aee493a605",
            "Image": "rhel",
            "Labels": {
                "Architecture": "x86_64",
                "Build_Host": "rcm-img04.build.eng.bos.redhat.com",
                "Name": "rhel-server-docker",
                "Release": "4",
                "Vendor": "Red Hat, Inc.",
                "Version": "7.1"
    ...
```

docker inspect --format: このコマンドを使用すると、コンテナーから情報の特定の部分を取得することができます。情報は階層的に保存されます。コンテナーの IP アドレス (NetworkSettings 下の IPAddress) を表示するには、**--format** オプションとコンテナーの ID を使用します。以下が例になります。

```
# docker inspect --format='{{.NetworkSettings.IPAddress}}' mybash
172.17.0.2
```

その他にも、.Path (コンテナーで実行するコマンドを表示する)、.Args (コマンドへの引数)、.Config.ExposedPorts (コンテナーから表示される TCP または UDP ポート)、.State.Pid (コンテナーのプロセス ID を表示する)、.HostConfig.PortBindings (コンテナーからホストへのポートマッピング) などを調べることができます。以下は、.State.Pid および .HostConfig.PortBindings の例になります。

```
# docker inspect --format='{{.State.Pid}}' mybash
5007
# docker inspect --format='{{.HostConfig.PortBindings}}' mybash
map[8000/tcp:[map[HostIp: HostPort:8000]]]
```

実行中の Docker コンテナー内の調査

実行中の Docker コンテナー内で調査を行うには、**docker exec** コマンドを使用することができます。**docker exec** を使用すると、コマンド (/bin/bash など) を実行して、実行中の Docker コンテナープロセスに入ってコンテナーを調べることができます。

コンテナーが目的のアプリケーションを実行している間にコンテナーを調べることができるため、コンテナーを bash シェルで起動するのではなく、**docker exec** を使用します。目的のタスクを実行中のコンテナーに割り当てると、コンテナーのアクティビティーを中断せずに、コンテナーの実際の機能をより詳細に確認できます。

以下の例では、**docker exec** を使用して、myrhel_httpd という名前の実行中のコンテナーを調べ、そのコンテナーの内部を調べます。

1. **コンテナーの起動**: 「Dockerfile からのイメージの構築」に記載されている myrhel_httpd コンテナーや、調査したいその他の Docker コンテナーを起動します。**docker ps** と入力し、コンテナーが実行されていることを確認します。

   ```
   # docker ps
   CONTAINER ID   IMAGE              COMMAND               CREATED
   STATUS        PORTS                NAMES
   1cd6aabf33d9   rhel_httpd:latest  "/usr/sbin/httpd -DF  6 minutes
   ago   Up 6 minutes  0.0.0.0:80->80/tcp   myrhel_httpd
   ```

2. **docker exec でコンテナーに入る**: コンテナー ID または名前を使用して bash シェルを開き、実行中のコンテナーにアクセスします。次に、以下のようにコンテナーの属性を調べることができます。

   ```
   # docker exec -it myrhel_httpd /bin/bash
   [root@1cd6aabf33d9 /]# cat /etc/redhat-release
   Red Hat Enterprise Linux Server release 7.1 (Maipo)
   [root@1cd6aabf33d9 /]# ps -ef
   UID        PID  PPID  C STIME TTY          TIME CMD
   root         1     0  0 08:41 ?        00:00:00 /usr/sbin/httpd
   -DFOREGROUND
   apache       7     1  0 08:41 ?        00:00:00 /usr/sbin/httpd
   -DFOREGROUND
   ...
   root        12     0  0 08:54 ?        00:00:00 /bin/bash
   root        35    12  0 08:57 ?        00:00:00 ps -ef
   [root@1cd6aabf33d9 /]# df -h
   Filesystem                          Size  Used Avail Use%
   Mounted on
   /dev/mapper/docker-253:0-540464...  9.8G  414M  8.8G   5%   /
   tmpfs                               7.9G     0  7.9G   0%
   ```

```
/dev
shm                                  64M      0   64M   0%
/dev/shm
/dev/mapper/rhel_unused-root        137G   45G   92G  33%
/etc/hosts
tmpfs                                7.9G  8.0K  7.9G   1%
/run
tmpfs                                7.9G  184K  7.9G   1%
/run/secrets
tmpfs                                7.9G     0  7.9G   0%
/proc/kcore
[root@1cd6aabf33d9 /]# uname -r
3.10.0-229.1.2.el7.x86_64
[root@1cd6aabf33d9 /]# rpm -qa | less
redhat-release-server-7.1-1.el7.x86_64
basesystem-10.0-7.el7.noarch
nss-softokn-freebl-3.16.2.3-9.el7.x86_64
...
bash-4.2# free -m
              total        used        free      shared
buff/cache   available
Mem:          16046        6759         641          20
8645        8948
Swap:          8071           2        8069
[root@1cd6aabf33d9 /]# ip addr show eth0
44: eth0:  mtu 1500 qdisc pfifo_fast state UP
    link/ether 92:b1:31:b2:79:69 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.14/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::90b1:31ff:feb2:7969/64 scope link
       valid_lft forever preferred_lft forever
[root@1cd6aabf33d9 /]# exit
```

(コンテナー内で実行中の) bash シェルからこのコマンドを実行すると、以下のことを確認できます。コンテナーは、RHEL Server リリース 7.1 システムを保持しています。プロセステーブル (ps -ef) は、httpd コマンドが (他の 5 つの httpd プロセスの後に続く) プロセス ID 1、/bin/bash が PID 12、および ps -ef が PID 35 であることを示しています。ホストのプロセステーブルで実行しているプロセスはコンテナー内では確認できません。コンテナーのファイルシステムは、9.8G の領域が利用できる root ファイルシステムで、414M を使用しています。

コンテナーでは他のカーネルは実行していません (uname -r はホストシステムのカーネル 3.10.0-229.1.2.el7.x86_64 を示します)。rpm -qa コマンドを使うと、コンテナーに含まれる RPM パッケージが表示されます。つまり、コンテナーには RPM データベースがあります。メモリーを表示 (free -m) すると、ホストで利用できるメモリーが表示されます (ただし、cgroups を使用してコンテナーが実際に使用できるものを制限できます)。コンテナーの IP アドレス (172.17.0.14/16) は、DHCP 経由でホストシステムからコンテナーに割り当てられます。この場合、ホストシステムには、IP アドレスが 172.17.42.1/16 の docker0 という名前のインターフェースがあります。

コンテナーの起動および停止

実行したコンテナーを実行しても、それを削除しない場合 (--rm)、そのコンテナーはローカルシステムに保存され、再度実行できます。実行した後に削除しなかったコンテナーを起動するには、**start** オプションを使用します。実行中のコンテナーを停止するには、**stop** オプションを使用します。

コンテナーの起動: docker コンテナーをインタラクティブに実行する必要がない場合は、start オプションとコンテナー ID (または名前) が必要になります。

```
# docker start myrhel_httpd
myrhel_httpd
```

コンテナーをローカルシェルから起動して使用するには、-a (attach) および -i (interactive) オプションを使用します。bash シェルが起動したら、コンテナー内で必要なコマンドを実行します。exit を実行すると、シェルを kill してコンテナーを停止できます。

```
# docker start -a -i agitated_hopper
bash-4.2# exit
```

コンテナーの停止: 端末のセッションに割り当てられていない実行中のコンテナーを停止するには、stop オプションとコンテナー ID または番号を使用します。以下が例になります。

```
# docker stop myrhel_httpd
myrhel_httpd
```

stop オプションは SIGTERM 信号を送り、実行中のコンテナーを停止します。しばらくの間 (デフォルトでは 10 秒) コンテナーが停止しないと、docker は SIGKILL 信号を送ります。さらに、**docker kill** コマンドを使用してコンテナーを kill (SIGKILL) するか、または別の信号をコンテナーに送信することもできます。以下は、SIGHUP 信号をコンテナーに送る例です (アプリケーションがサポートしている場合、SIGHUP によりアプリケーションが設定ファイルを再読み込みを行います)。

```
# docker kill --signal="SIGHUP" myrhel_httpd
```

コンテナーの削除

システムに保存されているコンテナーの一覧を表示するには、**docker ps -a** コマンドを実行します。不要なコンテナーを削除するには、オプションのコンテナー ID または名前と共に **docker rm** コマンドを使用します。以下は例になります。

```
# docker rm goofy_wozniak
```

同じコマンドラインで複数のコンテナーを削除できます。

```
# docker rm clever_yonath furious_shockley drunk_newton
```

すべてのコンテナーを削除する場合は、以下のようなコマンドを使用してローカルシステムからすべてのコンテナー (イメージではない) を削除できます。このコマンドを実行すると、すべてのコンテナーが削除されることに注意してください。

```
# docker rm $(docker ps -a -q)
```

7.5.6. Docker イメージの作成

ここまでの手順で、既存の docker コンテナーイメージを取得し、各種の方法でそれらを使用できるようになりました。必要なコンテナーをより少ない手順で実行するプロセスを用意するには、Docker イメージを始めから作成するか、既存のイメージを別のコンテンツまたは設定と組み合わせて実行したコンテナーから作成できます。

コンテナーからのイメージの作成

以下の手順では、既存のイメージ (rhel:latest) と選択しているパッケージセット(例: Apache Web サーバーの httpd) から新規イメージを作成する方法について説明します。

注意: 現在の Red Hat Docker リリースの場合、Red Hat から取得したデフォルトの RHEL 7 Docker イメージは、ホストシステムから利用できる RHEL 7 エンタイトルメントで利用できます。そのため、Docker ホストが適切にサブスクライブされていて、コンテナーで必要なソフトウェアを取得するためのリポジトリーが有効にされている限り (さらに Docker ホストからインターネットにアクセスできる場合)、RHEL 7 ソフトウェアリポジトリーからパッケージをインストールできます。

1. **新規コンテナーへの httpd のインストール**: Red Hat カスタマーポータルからローカルシステムに **rhel** イメージをロードし、Red Hat のサブスクリプション管理を使用してホストを適切にサブスクライブした場合は、次のコマンドを使用して以下を実行できます。

   - イメージをベースイメージとして使用する。
   - 現在インストールしているパッケージの最新バージョンを取得する (アップデート)。
   - httpd パッケージ (およびすべての依存パッケージ) をインストールする。
   - yum の一時キャッシュファイルをすべて削除する。

   ```
   # docker run -i rhel:latest /bin/bash -c "yum clean all;  \
       yum update -y; yum install -y httpd; yum clean all"
   ```

2. **新規イメージのコミット**: 新規コンテナーの ID または名前 (**docker ps -l**) を取得して、そのコンテナーをローカルリポジトリーにコミットします。コンテナーを新規イメージにコミットすると、コメント (-m) および作者名 (-a) を追加できます。さらに、イメージ名 (rhel_httpd) を新たに設定できます。次に **docker images** と入力して、新規イメージをイメージの一覧で確認します。

   ```
   # docker ps -l
   CONTAINER ID  IMAGE          COMMAND              CREATED
   STATUS                     PORTS  NAMES
   f6832df8da0a  redhat/rhel7:0 /bin/bash -c 'yum cl About a minute
   ago  Exited (0) 13 seconds ago        backstabbing_ptolemy4
   # docker commit -m "RHEL with httpd" -a "Chris Negus"
   f6832df8da0a rhel_httpd
   630bd3ff318b8a5a63f1830e9902fec9a4ab9eade7238835fa6b7338edc988ac
   # docker images
   REPOSITORY   TAG     IMAGE ID      CREATED         VIRTUAL SIZE
   rhel_httpd   latest  630bd3ff318b  27 seconds ago  170.8 MB
   redhat/rhel  latest  e1f5733f050b  4 weeks ago     140.2 MB
   ```

3. **新規イメージからのコンテナーの実行**: 作成したばかりのイメージを使用して、以下の **docker run** コマンドを実行し、インストールした Web サーバー (httpd) を起動します。以下は例になります。

   ```
   # docker run -d -p 8080:80 rhel_httpd:latest \
           /usr/sbin/httpd -DFOREGROUND
   ```

   この例では、Apache Web サーバー (httpd) はコンテナーのポート 80 で listen していて、このコンテナーはホストのポート 8080 にマップされています。

4. **コンテナーが実行中であることの確認**: 起動した httpd サーバーが利用可能であることを確認するには、そのサーバーからファイルを取得してみます。ホストからアドレスが http://localhost:8080 の Web ブラウザーを開くか、または curl などのコマンドラインユーティリティーを使用して httpd サーバーにアクセスします。

   ```
   # curl http://localhost:8080
   ```

Dockerfile からのイメージの構築

これまでの手順でイメージやコンテナーをコマンドラインから作成する方法について説明したので、このセクションではより永続的な方法でコンテナーを構築します。Docker フォーマットのコンテナーを作成するには、Dockerfile ファイルからコンテナーイメージを構築する方法が、実行中のコンテナーを修正し、それらのコンテナーをイメージにコミットすることよりもはるかに推奨される方法になります。

以下の手順では、前述の数多くの機能を含む Dockerfile ファイルを作成します。

- ベースイメージの選択
- Apache Web サーバー (httpd) に必要なパッケージのインストール
- サーバーのポート (TCP ポート 80) のホストの別のポート (TCP ポート 8080) へのマッピング
- Web サーバーの起動

RHEL 7 で Docker 開発環境を設定する機能は多数ありますが、ここでは、独自の docker コンテナーを構築する際に注意する必要がある点についていくつか紹介します。

- **エンタイトルメント**: コンテナーに関連する Red Hat エンタイトルメント関連の問題についていくつか紹介します。
  - Red Hat Subscription Manager を使用して Docker ホストシステムのサブスクライブを行う場合、そのホストに Docker イメージを構築する際に、ビルド環境はホストで有効にしたのと同じ Red Hat ソフトウェアリポジトリーに自動的にアクセスできます。
  - コンテナーを構築する際にさらに多くのリポジトリーを利用できるようにするには、ホスト上またはコンテナー内でそれらのリポジトリーを有効にします。
  - subscription-manager コマンドはコンテナーでサポートされていないため、/etc/yum.repos.d/redhat.repo ファイル内で repo を有効にすると、リポジトリーを有効または無効にできます。また、yum-utils パッケージをコンテナーにインストールして yum-config-manager コマンドを実行できます。
  - RHEL 6 コンテナーを RHEL 7 ホストに構築する場合は、ホストで有効にされている RHEL 6 バージョンのリポジトリーを自動的に選択します。
  - コンテナー内の Red Hat エンタイトルメントの詳細は、Docker Entitlements ソリューションを参照してください。
- **アップデート**: Red Hat Enterprise Linux の Docker コンテナーには、アップデートしたソフトウェアパッケージが自動的に含まれません。Docker イメージを再構築して、パッケージを最新にしたり、重要なアップデートが必要な場合にすぐに再構築したりする必要がある場合があります。たとえば、以下の Dockerfile の例で示される "RUN yum update -y" 行を使用して、Docker イメージが再構築されるたびにパッケージをアップデートできます。
- **イメージ**: デフォルトで、docker ビルドはローカルキャッシュから特定するベースイメージの最新バージョンを使用します。新規イメージを構築する前に、リモート Docker レジストリーからイメージの最新バージョンを取得 (docker pull コマンドを使用) する必要がある場合があります。イメージの特定のインスタンスが必要な場合はタグを指定します。たとえば、イメージ "centos" を指定すると、centos:latest イメージが取得されます。CentOS 6 のイメージが必要な場合は、centos:centos6 イメージを取得する必要があります。

    - **プロジェクトディレクトリーの作成**: docker および docker-registry サービスを実行中のホストシステムで、プロジェクトのディレクトリーを作成します。

```
# mkdir -p httpd-project
# cd httpd-project
```

- **Dockerfile ファイルの作成**: テキストエディターを使用して Dockerfile という名前のファイルを開きます (**vim Dockerfile** など)。ホストの RHEL 7 システムを登録し、サブスクライブを行なった場合は、以下が httpd サーバーの Docker コンテナーを構築する際の Dockerfile ファイルの例になります。

```
# My cool Docker image
# Version 1

# If you loaded redhat-rhel-server-7.0-x86_64 to your local
registry, uncomment this FROM line instead:
# FROM registry.access.redhat.com/rhel
# Pull the rhel image from the local registry
FROM registry.access.redhat.com/rhel

MAINTAINER Chris Negus

# Update image
RUN yum update -y
# Add httpd package. procps and iproute are only added to
investigate the image later.
RUN yum install httpd procps iproute -y
RUN echo container.example.com > /etc/hostname

# Create an index.html file
RUN bash -c 'echo "Your Web server test is successful." >>
/var/www/html/index.html'
```

- **Dockerfile 構文の作成 (オプション)**: Red Hat は Red Hat カスタマーポータルで Docker file ファイルを確認するツールを提供しています。Dockerfile ファイルを構築する前に Linter for Dockerfile ページに移動してそのファイルを確認することができます。

- **イメージの構築**: Dockerfile ファイルからイメージを構築するには、build オプションを使用して Dockerfile ファイルの場所を指定する必要があります (ここでは現行ディレクトリーを示す "." を使用)。

  注意: docker build で --no-cache オプションを使用することを検討してください。--no-cache を使用すると、各ビルド層のキャッシュを取得しないで、過剰なディスク領域の使用を防ぐことができます。

```
# docker build -t rhel_httpd .
Uploading context  2.56 kB
Uploading context
Step 0 : FROM registry.access.redhat.com/rhel
 ---> f5f7ddddef7d
Step 1 : MAINTAINER Chris Negus
 ---> Running in 3c605e879c72
 ---> 77828ebe8f6f
Removing intermediate container 3c605e879c72
Step 2 : RUN yum update -y
 ---> Running in 9f45bb262dc6
...
```

```
 ---> Running in f44ea9eb6155
 ---> 6a532e340ccf
Removing intermediate container f44ea9eb6155
Successfully built 6a532e340ccf
```

- **イメージ内での httpd サーバーの実行**: 以下のコマンドを実行すると、構築したイメージ (この例では rhel_httpd) から httpd サーバーを実行できます。

```
# docker run -d -t --name=myrhel_httpd \
        -p 80:80 -i rhel_httpd:latest \
        /usr/sbin/httpd -DFOREGROUND
```

- **サーバーが実行中であることの確認**: ホストの別の端末で以下を入力し、httpd サーバーにアクセスできることを確認します。

```
# netstat -tupln | grep 80
tcp6       0      0 :::80       :::*      LISTEN
26137/docker-proxy
# curl localhost:80
Your Web server test is successful.
```

イメージのタグ付け

何が含まれているかを判別しやすくするために、名前をイメージに追加できます。**docker tag** コマンドを使用すると、基本的に複数の部分で構成されるイメージにエイリアスが追加されます。これには以下が含まれます。

registryhost/username/NAME:tag

NAME だけを追加することもできます。以下は例になります。

```
# docker tag 474ff279782b myrhel7
```

ここで、**rhel7** イメージのイメージ ID は 474ff279782b です。**docker tag** を使用すると、名前 **myrhel7** をイメージ ID に割り当てることもできます。そのため、このコンテナーを名前 (rhel7 または myrhel7) またはイメージ ID で実行できます。:tag を名前に追加しないと、:latest がタグとして割り当てられます。以下のようにタグを 7.1 に設定できます。

```
# docker tag 474ff279782b myrhel7:7.1
```

オプションで、ユーザー名やレジストリー名を名前の前に追加できます。ユーザー名は、実際はリポジトリーを所有するユーザーアカウントに関連する Docker.io のリポジトリーです。イメージにレジストリー名を使ってタグ付けする方法は「Docker レジストリーの外からイメージを取得する」を参照してください。以下は、ユーザー名を追加する例です。

```
# docker tag 474ff279782b cnegus/myrhel7
# docker images | grep 474ff279782b
rhel7           latest  474ff279782b  7 months ago  139.6 MB
myrhel7         latest  474ff279782b  7 months ago  139.6 MB
myrhel7         7.1     474ff279782b  7 months ago  139.6 MB
cnegus/myrhel7  latest  474ff279782b  7 months ago  139.6 MB
```

ここでは、1 つのイメージ ID に割り当てられたイメージ名がすべて表示されます。

イメージの保存およびインポート

作成した Docker イメージを保存する場合は、docker save を実行すると tarball にイメージを保存できます。tarball の作成後にはイメージを再ロードし、再利用することができるため、それを保存したり、他のユーザーに送信したりすることができます。以下は、イメージを tarball として保存する例です。

```
# docker save -o myrhel7.tar myrhel7:latest
```

myrhel7.tar ファイルは現在のディレクトリーに保存されます。後に tarball をコンテナーイメージとして再利用する場合は、以下のようにこれを別の docker 環境にインポートできます。

```
# cat myrhel7.tar | docker import - cnegus/myrhel7
```

イメージの削除

システムにあるイメージの一覧を確認するには、**docker images** コマンドを実行します。不要になったイメージを削除するには、**docker rmi** コマンドにオプションとしてイメージ ID または名前を追加して実行します (イメージを削除する前に、そのイメージを使用しているコンテナーを停止する必要があります)。以下は例になります。

```
# docker rmi rhel
```

同じコマンドラインで複数のイメージを削除することができます。

```
# docker rmi rhel fedora
```

すべてのイメージを削除する必要がある場合、以下のようなコマンドを使用してローカルレジストリーからすべてのイメージを削除できます (このコマンドを実行すると、すべてのイメージが削除されます)。

```
# docker rmi $(docker images -a -q)
```

7.6. まとめ

現在、docker と docker-registry サービスが動作している Red Hat Docker をインストールことができます。コンテナーの実行方法や独自のイメージを構築する方法を確認できるように、1 つ以上の Docker メージを使用することができます。

第8章 DOCKER フォーマットコンテナーを使用したストレージの管理

8.1. 概要

実稼働環境で数多くのコンテナーを実行するには、多くのストレージ領域が必要になります。さらに、コンテナーを作成し、実行するには、基礎となるストレージドライバーが最も効率のよいオプションを使用できるように設定される必要があります。Docker のデフォルトのストレージオプションはシステム間で異なり、場合によっては変更する必要がある場合もあります。RHEL のデフォルトインストールはループバックデバイスを使用し、RHEL Atomic Host ではインストール時に LVM シンプールが作成されます。ただし、ループバックオプションを実稼働システムで使用することは推奨されません。計画段階では、以下を確認してください。

1) direct-lvm を実行しており、LVM シンプールがセットアップされている。これは **docker-storage-setup** ユーティリティーを使用して実行できます。

2) インストール時に十分な空き領域を割り当てているか、または外部ストレージがシステムに割り当てられるように計画する。

本書では、領域が不足した場合にストレージを拡張する手順についても記載しています。これらの手順の一部は破壊的な操作を伴うため、前もってよく計画することをお勧めします。ご使用のシステムに関連した手順を使用して、環境のセットアップを行ってください。

8.2. DOCKER-STORAGE-SETUP の使用

docker-storage-setup ユーティリティーは **docker** パッケージと共にインストールされ、direct LVM ストレージのセットアップに役立ちます。

docker が起動すると、**docker-storage-setup** デーモンを自動的に起動します。デフォルトでは、**docker-storage-setup** は root 論理ボリュームを含むボリュームグループに空き領域を探そうとし、LVM シンプールのセットアップを試行します。ボリュームグループに空き領域がない場合、**docker-storage-setup** は LVM シンプールをセットアップせず、ループバックデバイスにフォールバックします。

docker-storage-setup のデフォルト動作は **/usr/lib/docker-storage-setup/docker-storage-setup** 設定ファイルによって制御されます。新しい値を使用してファイル **/etc/sysconfig/docker-storage-setup** を作成して、オプションを上書きできます。

docker-storage-setup では、シンプールをセットアップするために空き領域のある場所を認識している必要があります。以下は、docker-storage-setup が LVM シンプールをセットアップできるようにシステムを設定するいくつかの方法です。

詳細は、**man docker-storage-setup(1)** を参照してください。(man ページは RHEL Atomic ではデフォルトで利用できず、RHEL Atomic Tools コンテナーをダウンロードしておく必要があることに注意してください。)

8.2.1. root ボリュームを含むボリュームグループの LVM シンプール

デフォルトでは、docker-storage-setup は root ボリュームグループの空き領域を探し、LVM シンプールを作成します。そのため、インストール時に root ボリュームグループに空き領域を残すと、docker の起動時にシンプールが自動的にセットアップされ、使用されます。

8.2.2. ユーザー指定のボリュームグループの LVM シンプール

docker-storage-setup は、シンプールを作成するために特定のボリュームグループを使用するように設定できます。

```
# echo VG=docker-vg >> /etc/sysconfig/docker-storage-setup
# systemctl start docker
```

8.2.3. ユーザー指定のブロックデバイスでのボリュームグループおよび LVM シンプールのセットアップ

1 つまたは複数のデバイスを **/etc/sysconfig/docker-storage-setup** ファイルに指定できます。docker-storage-setup は、docker サービスが使用するボリュームグループと LVM シンプールを作成します。

```
# echo DEVS=/dev/vdb >> /etc/sysconfig/docker-storage-setup
# systemctl start docker
```

8.3. RED HAT ENTERPRISE LINUX におけるストレージの管理

Red Hat Enterprise Linux では、デフォルトでは root ボリュームグループに空き領域はありません。そのため、docker-storage-setup が空き領域を見つけられるようにするための操作が必要です。

簡単な方法として、インストール時に root を含むボリュームグループにいくらかの空き領域を残すことができます。以下のセクションでは空き領域を残す方法を説明します。

8.3.1. インストール時に root ボリュームグループに空き領域を残す方法

インストール時に root ボリュームグループに空き領域を確保する 2 つの方法があります。インタラクティブなグラフィカルインストールユーティリティーの Anaconda を使用するか、またはインストールを制御するキックスタートファイルを準備します。

8.3.1.1. GUI インストール

1. グラフィカルインストールを開始します。「Installation Destination」画面に移動し、「Other Storage Options」から「I will configure partitioning」を選択して「Done」をクリックします。
2. マウントポイントを作成するようにプロンプトが出される「Manual Partitioning」画面で、「Click here to create them automatically」を選択します。これにより、デフォルトのパーティショニングスキームが作成されます。
3. root パーティション (*/*) を選択します。これにより、「Desired Capacity」入力フィールドが表示されます。
4. その容量を縮小し、root ボリュームグループに空き領域を確保します。
5. デフォルトで、root LV を持つボリュームグループにはユーザーが作成したボリュームに対応する十分な大きさがあります。また、ディスク上のすべての空き領域は空き領域のままとなり、そのボリュームグループの一部にはなりません。この設定を変更するには、「Modify」をクリックし、「Size policy」を選択して「As large as possible」に設定し、「Save」をクリックします。これにより、ディスク上の使用されていない領域はボリュームグループ内の空き領域になります。

6. 「Done」をクリックして、提案されるパーティション設定を受け入れます。
7. 「Begin Installation」をクリックします。

8.3.1.2. キックスタートインストール

キックスタートファイルでは、"volgroup" キックスタートオプションを "--reserved-percent" および "--reserved-space" オプションと共に使用します。ここで、ボリュームグループ内でどの程度の領域を確保するかを指定できます。以下は、root LV に20% の空き領域を残すキックスタートファイルのセクション例です。

```
# Disk partitioning information
part /boot --size=500
part pv.1 --size=500 --grow
volgroup rhel --pesize=4096 pv.1 --reserved-percent=20
logvol /  --size=500 --grow --name=root --vgname=rhel
logvol swap --size=2048 --name=swap --vgname=rhel
```

8.4. RED HAT ENTERPRISE LINUX ATOMIC HOST におけるストレージの管理

RHEL Atomic Host では、root ボリュームのサイズは 3GB です。root ボリュームグループには空き領域があり、その 60% は LVM シンプールのセットアップのために **docker-storage-setup** によって使用されます。残りの領域は空き領域となり、root ボリュームを拡張したり、シンプールを作成したりするために使用できます。

デフォルトのパーティションが設定されている RHEL Atomic Host では、**docker-storage-setup** サービスはコンテナーイメージで使用される LVM シンプールを作成します。インストール時に、インストールプログラムはデフォルトで 3GB の **root** 論理ボリュームを作成します。次は起動時に、docker-storage-setup サービスが、残りの領域の 60% を占める **docker-pool** という LVM シンプールを自動的にセットアップします。残りの領域は **root** または **docker-pool** を拡張するために使用できます。起動時に、docker-storage-setup は /etc/sysconfig/docker-storage ファイルを読み取り、使用されているストレージのタイプを判別し、Docker が LVM シンプールを利用できるようにこれを変更します。デフォルト値は、/etc/sysconfig/docker-storage-setup という名前のファイルを作成して上書きできます。これにより、起動時のサービスの動作が変更されます。このファイルを作成しない場合、LVM シンプールがデフォルトで作成されます。

デフォルトのパーティション設定でクラウドイメージからインストールされる Red Hat Enterprise Linux Atomic Host には、**atomicos** というボリュームグループと、そのグループの中に 2 つの論理ボリュームがあります。ボリュームグループの名前は、 Red Hat Enterprise Linux Atomic Host のイメージごとに異なります。ベアメタルおよび仮想インストールの場合、ボリュームグループにはホスト名に由来する名前が付きます。ホストの名前が付いていない場合、ボリュームグループは **rah** になります。ボリュームグループおよびそれらの中にある論理ボリュームのプロパティーはすべてのイメージ間で同一のプロパティーになります。

lvs コマンドを実行して、システムの論理ボリュームを一覧表示し、ボリュームグループ名を表示できます。

```
# lvs
LV                VG          Attr          LSize Pool Origin Data%
Meta%  Move Log Cpy%Sync Convert
docker-pool       atomicos    twi-aotz--    7.69g             14.36
2.56
root              atomicos    -wi-ao----    2.94g
```

1. **root パーティション** は root と呼ばれ、デフォルトでは 3GB です。root は以下を含む論理ボリュームです。
   - **/var** および **/etc** ディレクトリー。
   - OSTree バージョンを含む **/ostree/repo**。
   - 一時データまたは **docker ボリューム** などのコンテナーのイメージデータが含まれる **/var/lib/docker/** ディレクトリー。**docker ボリューム** は、実行中のコンテナーがホストシステムに要求できるストレージのユニットです。このストレージのユニットは別のコンテナーで提供されるだけでなく、ホストから直接提供することもできます。Red Hat Enterprise Linux Atomic Host の場合、それらのボリュームは **/var/lib/docker/vfs/** の **root パーティション** に自動的に割り当てられます。
2. **コンテナーイメージパーティション** は **docker-pool** と呼ばれ、残りの領域の 60% を使用します。これは docker-storage-setup サービスによって LVM シンプールとしてフォーマットされます。このパーティションはコンテナーイメージを保存するために使用されます。**docker-pool** で使用される領域は docker-storage-setup サービスによって管理されます。レジストリーからコンテナーイメージを取得する場合など、イメージはこのパーティションの領域を占めます。コンテナーイメージは読み取り専用です。イメージがコンテナーとして起動すると、すべての書き込み (マウントされたボリュームまたは docker ボリュームに対するものを除く) はこの論理ボリュームに保存されます。

docker-pool で空き領域を監視し、領域が不足した状態で実行されないようにすることは非常に重要です。LVM シンプールが領域が不足すると、LVM シンプールの基礎となる XFS ファイルシステムが I/O エラーに対して再試行を無制限に繰り返すため、障害が発生します。LVM2 ツールはユーザー設定に基づいてシンプールを監視し、これを拡張するための機能を提供します。詳細は、**lvmthin(7)** man ページの **Automatically extend thin pool LV and Data space exhaustion** セクションを参照してください。デフォルトでは、docker-storage-setup は自動拡張用にシンプールを設定します。つまり、プールが一杯になると、自動的にボリュームグループ内の利用可能な空き領域が拡張し、これが使用されます。ボリュームグループが一杯になると、自動拡張用の領域がなくなるため、領域を確保するために不要な古いコンテナーを破棄するなど措置を事前に取ることができます。または、追加のストレージがシステムに追加されるまで、コンテナーイメージの作成または変更を停止することもできます。

- **/etc/sysconfig/docker** - ユーザーによって設定されます。
- **/etc/sysconfig/docker-storage** - プログラムによって設定されますが、ユーザーが編集することもできます (docker-storage-setup を無効にする必要があります)。
- **/etc/sysconfig/docker-storage-setup** - ユーザーによって設定されますが、RHEL Atomic Host でのみ利用できます。

8.4.1. インストール時の root パーティションのデフォルトサイズの変更

デフォルトの **root パーティション** のサイズを変更するには、ご使用のインストール用に以下の方法を使用します。

- **Anaconda**: 「Installation Destination」画面に移動して、「Other Storage Options」から「I will configure partitioning」を選択して「Done」をクリックします。これにより、「Manual Partitioning」画面に移動し、マウントポイントを作成するようにプロンプトが出されます。「Click here to create them automatically」をクリックすると boot、root、および swap パーティションが提供されます (この時点ではこれらのパーティションのみがあり、**docker-pool** は docker-storage-setup サービスで後で作成されます)。root パーティション (/) を選択し、「Desired Capacity」入力フィールドに新たな値を入力します。インストールが終了したら、システムはカスタム設定で起動します。

- **キックスタート**: キックスタートファイルの %post セクションで、/etc/sysconfig/docker-storage-setup ファイル (自動作成される) へのパスを指定し、コマンドの後に必要なオプションを指定します。構文は以下のようになります。

```
%post
cat > /etc/sysconfig/docker-storage-setup << EOF
ROOT_SIZE=6G
EOF
%end
```

- **cloud-init**: user-data ファイルの **write_files** ディレクティブは、上記のキックスタートの例のように /etc/sysconfig/docker-storage-setup ファイルをセットアップするために使用されます。この例の user-data ファイルは **cloud-user** のパスワードを "atomic" に設定し、root パーティションがデフォルトの 3GB ではなく 6GB になるように設定します。

```
#cloud-config
password: atomic
write_files:
  - path: /etc/sysconfig/docker-storage-setup
    permissions: 0644
    owner: root
    content: |
      ROOT_SIZE=6G
```

8.4.2. インストール後の **root** パーティションのサイズの変更

コンテナーイメージを、/var/lib/docker/ で領域を必要とする **コンテナーイメージパーティション** に追加する場合、イメージは現時点で **root パーティション** で利用可能なサイズよりも大きな領域を要求できます。コンテナーイメージは、データベースサーバーからのデータなど、コンテナーに保存できないデータがある場合に docker ボリュームを要求できます。**root** の領域が不足している場合は、以下の 3 つのオプションを選択できます。

- root パーティションを拡張して、ボリュームグループ内の空き領域を使用する。
- 新規ストレージをホストに追加し、root パーティションを拡張する。
- root パーティションを拡張し、コンテナーイメージパーティションを縮小する。

8.4.2.1. **root** パーティションを拡張してボリュームグループの空き領域を使用する方法

ボリュームグループに空き領域がある場合、root ボリュームを拡張して、空き領域の一部およびすべてを使用して root パーティションを拡張することができます。

```
# lvextend -r -L +3GB /dev/atomicos/root
```

8.4.2.2. 追加ストレージをホストに追加し、**root** パーティションを拡張する方法

このオプションに破壊的は操作は伴わず、追加のストレージを **root パーティション** に追加し、これを使用することができます。これには、新規のディスクデバイスを使用して新規の物理ボリュームを作成することが必要になり (この場合は /dev/sdb)、これを **atomicos** ボリュームグループに追加してから **root パーティション** 論理ボリュームを拡張します。このタスクについては、docker デーモンおよび docker-storage-setup サービスを停止する必要があります。以下のコマンドを使用します。

```
# systemctl stop docker docker-storage-setup
# pvcreate /dev/sdb
# vgextend atomicos /dev/sdb
# lvextend -r -L +3GB /dev/atomicos/root
# systemctl start docker docker-storage-setup
```

8.4.2.3. ストレージを追加せずに root パーティションを拡張する方法

このオプションには、**コンテナーイメージパーティション** を破棄するような破壊的な操作を伴います。追加のストレージを **root パーティション** に追加できない場合、このパーティションを拡張することができます。**root パーティション** を拡張するには、**コンテナーイメージパーティション** を縮小しなければなりません。ただし、LVM は縮小するシンプロビジョニングされた論理ボリュームをサポートしません。

したがって、実行中のすべてのコンテナーを停止し、**コンテナーイメージパーティション** を破棄し、**root パーティション** を拡張する必要があります。**docker-storage-setup** は、再起動時に残りの領域を **コンテナーイメージパーティション** に再度割り当てます。以下のコマンドを使用します。

```
# systemctl stop docker docker-storage-setup
# rm -rf /var/lib/docker/*
# lvremove atomicos/docker-pool
# lvextend -L +3GB /dev/atomicos/root
# systemctl start docker-storage-setup
# systemctl start docker
```

この時点で、すべてのコンテナーイメージを再びダウンロードする必要があります。

8.5. DOCKER ストレージ設定の変更

Docker のストレージ設定を変更する場合、**/var/lib/docker** ディレクトリーを必ず削除してください。このディレクトリーには、新しい設定では無効となる古いイメージ、コンテナー、およびボリュームのメタデータが含まれます。ストレージ設定の変更が必要になる可能性のある例には、ループデバイスの使用から LVM シンプールに切り替える場合や、あるシンプールから別のシンプールに切り替える場合が含まれます。後者の場合、古いシンプールを削除する必要があります。

```
# systemctl stop docker docker-storage-setup
# rm /etc/sysconfig/docker-storage-setup
# lvremove docker/docker-pool
# rm -rf /var/lib/docker/
# systemctl start docker
```

8.6. OVERLAY グラフドライバー

overlay グラフドライバーは、スナップショットボリューム間のページキャッシュの共有を特長とする COW (copy-on-write) 方式の union ファイルシステムを使用します。LVM シンプールと同様に、OverlayFS はイメージ層の効率的な保存をサポートします。ただし、LVM シンプールと比較すると、OverlayFS を使用した場合、コンテナーの作成および破棄に使用されるメモリーはより少なくなるため、パフォーマンスがよくなります。

警告

OverlayFS は POSIX に準拠しておらず (一部のファイルシステムのセマンティクスの一部は ext4 および XFS などの標準ファイルシステムとは異なる)、SELinux をまだサポートしていないことに注意してください。したがって、OverlayFS を Docker で有効にする前に、アプリケーションが OverlayFS で機能することを確認してください。Docker で OverlayFS を使用する方法については、『Red Hat Enterprise Linux 7.2 リリースノート』の 第 17 章 ファイルシステム を参照してください。

Docker の **overlay** グラフドライバーを有効にする一般的な方法は、SELinux を無効にし、**/etc/sysconfig/docker-storage-setup** で **overlay** を指定する方法です。

重要

ストレージのバックエンドの変更には破壊的な操作が伴います。開始する前に、必ず **docker save** でイメージをバックアップしてください。その後に、**docker load** でバックアップからイメージを復元することができます。

Docker を停止し、現行のストレージを削除します。

```
# systemctl stop docker docker-storage-setup
# rm -rf /var/lib/docker/
```

/etc/sysconfig/docker の **OPTIONS** 変数からオプションの **--selinux-enabled** を削除して SELinux を無効にします。

```
# sed -i '/OPTIONS=/s/--selinux-enabled//' /etc/sysconfig/docker
```

/etc/sysconfig/docker-storage-setup で **STORAGE_DRIVER** を **overlay** に設定します。

```
STORAGE_DRIVER=overlay
```

docker-storage-setup を再起動してから docker を再起動します。

```
# systemctl start docker-storage-setup
# systemctl start docker
```

- **キックスタート**

  キックスタートインストールの場合、**%post** セクションで以下のコマンドを使用します。

  ```
  %post
  sed -i '/OPTIONS=/s/--selinux-enabled//' /etc/sysconfig/docker
  echo "STORAGE_DRIVER=overlay" >> /etc/sysconfig/docker-storage-setup
  %end
  ```

- **cloud-init**

  cloud-init インストールの場合、**user-data** ファイルに以下のスニペットを組み込みます。

```
runcmd:
  - sed -i '/OPTIONS=/s/--selinux-enabled//' /etc/sysconfig/docker
  - echo "STORAGE_DRIVER=overlay" >> /etc/sysconfig/docker-storage-
setup
```

8.7. ストレージについての追加情報

- 『LVM 管理ガイド』の「シンプロビジョニングされた論理ボリューム (シンボリューム)」セクションでは、LVM シンプロビジョニングについて詳細に説明しています。
- 『Red Hat Enterprise Linux 7 ストレージ管理ガイド』は、Red Hat Enterprise Linux 7 にストレージを追加する方法についての情報を提供しています。

第9章 SYSTEMD を使用したコンテナーの起動

起動時にコンテナーを自動的に起動するには、まず **/etc/systemd/system/** ディレクトリーのユニット設定ファイルを作成して、コンテナーを systemd サービスとして設定します。たとえば、**/etc/systemd/system/redis-container.service** の内容は以下のようになります。

```
[Unit]
Description=Redis container
Author=Me
After=docker.service

[Service]
Restart=always
ExecStart=/usr/bin/docker start -a redis_server
ExecStop=/usr/bin/docker stop -t 2 redis_server

[Install]
WantedBy=local.target
```

ユニットファイルの作成後は、**systemctl enable** コマンドを使用してコンテナーを自動的に起動できます。

systemd でサービスを設定する方法についての詳細は、Red Hat Enterprise Linux 7 システム管理者のガイド の「systemd によるサービス管理」の章を参照してください。

第10章 スーパー特権コンテナーの実行

10.1. 概要

コンテナーは、独自の名前空間が含まれたビューを維持し、実行しているホストへのアクセスを制限するために設計されました。コンテナーには、デフォルトでは、ホストと異なるプロセステーブル、ネットワークインターフェース、ファイルシステム、および IPC 機能があります。ホストシステムやその他のコンテナーへのアクセスを制御するための機能や SELinux などの多くのセキュリティー機能がコンテナーに使用されています。コンテナーはホストのリソースを使用できますが、コンテナーから実行するコマンドは、ホストとの直接的なインターフェースの機能としては非常に制限されています。

ただし、一部のコンテナーはホストシステムの機能に直接アクセスし、監視し、それらの機能を変更することを目的に作成されています。これらは、**スーパー特権コンテナー** と呼ばれています。Red Hat Enterprise Linux Atomic Host (RHEL Atomic) の性質により、SPC (スーパー特権コンテナー) は RHEL Atomic Host の使用における重要な機能を提供しています。以下が例になります。

- RHEL Atomic Host はスリム化を目的としています。そのため、RHEL Atomic Host の管理またはトラブルシューティングを実行するために使用する数多くのツールはデフォルトでは組み込まれていません。
- Atomic Host では **yum** または **rpm** コマンドを使用してパッケージをインストールすることができないため、RHEL またはサードパーティーのツールを RHEL Atomic Host に追加する場合は、これらのツールをコンテナーに組み込むことが最も良い方法になります。
- SPC (スーパー特権コンテナー) を RHEL Atomic ホストに追加し、問題をトラブルシューティングし、不要になったリソースを削除して解放します。

Red Hat は、とくに RHEL Atomic Host で実行するための SPC (スーパー特権コンテナー) を複数用意しており、今後さらに追加していく予定です。これらには、以下が含まれます。

- **RHEL Atomic Tools コンテナーイメージ**: このコンテナーは、管理者のシェルと見なすことができます。多くのデバッグツール (strace、traceroute、sosreport など) と man ページがこのコンテナー内にあり、これらは管理者がホスト上の問題を診断するために使用できる可能性があります。
- **RHEL Atomic rsyslog コンテナーイメージ**: このコンテナーは rsyslogd サービスを実行したり、ログメッセージを集中型サーバーに送ったり、RHEL Atomic のログファイルを管理したりします。systemd-journald サービスは、rsyslog コンテナーをインストールしていなくても、RHEL Atomic Host のすべてのログデータを収集することに注意してください。
- **RHEL Atomic System Activity Data Collector (sadc) コンテナーイメージ**: このコンテナーは sysstat パッケージから sadc サービスを実行し、**sar** コマンドを実行することで RHEL Atomic システムが後で読み込むことのできるデータを継続的に収集できるようにします。

本書では、RHEL Atomic Tools コンテナーイメージのサンプルを使用して、スーパー特権コンテナーがどのように実行され、SPC (スーパー特権コンテナー) がホストの機能にどのようにアクセスするかを説明します。

10.2. 特権コンテナーの実行

docker コマンドを実行し、スーパー特権コンテナーとして実行する必要のあるすべてのオプションをすべて追加するには、長くて複雑なコマンドラインが必要です。そのため、コンテナーを実行する **atomic** コマンドを導入することで、このプロセスを単純化しました。以下のように **atomic** コマンドを実行します。

■

```
# atomic run rhel7/rhel-tools
[root@localhost /]#
```

これにより、複数のオプションを追加した docker コマンドを使用して rhel-tools コンテナーを作成し、起動します。これにより、RHEL Atomic Hoast でコンテナーを使用し、実行することがより簡単になります。実際の docker コマンドは以下のようになります。

```
docker run -it --name rhel-tools --privileged                          \
      --ipc=host --net=host --pid=host -e HOST=/host                   \
      -e NAME=rhel-tools -e IMAGE=rhel7/rhel-tools                     \
      -v /run:/run -v /var/log:/var/log                                \
      -v /etc/localtime:/etc/localtime -v /:/host rhel7/rhel-tools
```

スーパー特権コンテナーでどのオプションが実行されるかを理解することにより、ホスト上のリソースにアクセスする必要のある独自のコンテナーを実行する際にそれらのオプションをどのように使用できるかについてよりよく理解できるでしょう。以下でこれらのオプションを説明します。

- **-i -t**: 端末デバイス (**-t**) を開き、対話的に実行します (**-i**)。
- **--name** オプションはコンテナーの名前を設定します (この場合は rhel-tools)。
- **--privileged** オプションはセキュリティーの分離をオフにします。つまり、コンテナー内で root で実行されているプロセスは、コンテナー外で実行した場合と同様に RHEL Atomic Host にアクセスできることを意味します。
- **--ipc=host**、**--net=host**、および **--pid=host** フラグは、コンテナー内の ipc、net、および pid 名前空間をオフにします。これは、コンテナー内のプロセスが同じネットワークとプロセステーブルを認識し、IPC をホストプロセスと共有することを意味します。

コンテナーに環境変数を設定するオプションがいくつかあります (**-e**)。コンテナーを起動するときに開かれるシェルからこれらのオプションを参照できます (例: **echo $HOST**)。これらには以下が含まれます。

- **-e HOST=/host**: コンテナーにあるホストのファイルシステムの場所を設定します (つまり、ホストから / がマウントされる場所)。$HOST を任意のファイル名に追加すると、実行するコマンドは、コンテナー内ではなくホストのファイルにアクセスします。たとえば、コンテナーから **$HOST/etc/passwd** がホストの **/etc/passwd** ファイルにアクセスします。
- **-e NAME=rhel-tools**: コンテナーの名前を設定します (**docker ps** の実行時に表示されます)。
- **-e IMAGE=rhel7/rhel-tools**: イメージの名前を特定します (**docker images** の実行時に表示されます)。

(ホストからコンテナーに通常マウントされるもの以外に) ホストの複数のファイルおよびディレクトリーは、ホストのファイルシステムからコンテナーにマウントされます。これらには以下が含まれます。

- **-v /run:/run**: **-v /run:/run** オプションは、ホストから **/run** ディレクトリーを、コンテナー内の **/run** ディレクトリーにマウントします。これにより、コンテナー内のプロセスは、ホストの dbus サービスと通信し、systemd サービスに直接通信できるようになります。コンテナー内のプロセスは、docker デーモンとも通信することができます。
- **-v /var/log:/var/log**: コンテナー内でコマンドを実行して、ホストの **/var/log** ディレクトリーからログファイルの読み取りおよび書き込みを実行できるようにします。
- **-v /etc/localtime:/etc/localtime**: ホストシステムのタイムゾーンをコンテナーと共に使用します。
- **-v /:/host**: ホストから **/** を **/host** にマウントすることにより、コンテナー内のプロセスからホス

トのコンテンツを簡単に変更できるようになります。**touch /host/etc/passwd** を実行することにより、実際はホスト上の **/etc/passwd** ファイルに対して動作することになります。

最後の引数は、実行するイメージとして **rhel7/rhel-tools** を指定します。

RHEL ツールの特権コンテナーの場合にこれを実行すると、シェルが開いて、コンテナー内からコマンドを使用できるようになります。または、特定のコマンド (**sosreport** または **traceroute** など) を実行するために、**atomic** または **docker** コマンドラインの最後にオプションを追加することもできます。次のセクションでは、このコンテナーを調査する方法について説明します。

10.3. 特権コンテナーの名前空間について

基本的な Red Hat Enterprise 管理コマンドの多くは、それらがコンテナー内で実行されていることを認識できるように変更されています。たとえば、RHEL Tools コンテナー内で **sosreport** を実行すると、**/** ではなく、ファイルシステムの root として **/host** を使用していることを認識します。RHEL ツールコンテナー (または任意の特権付きコンテナー) から他のコマンドを実行する場合は、以下の点について特権コンテナーで実行される場合と動作が異なります。

10.3.1. 特権

特権コンテナーは、デフォルトではホストの root ユーザーとしてアプリケーションを実行します。コンテナーでは unconfined_t SELinux セキュリティーコンテキストで実行されるため、この機能が含まれます。

10.3.2. マウントテーブル

df および **mount** などのツールを使用してマウントされているファイルシステムを確認する場合、同じコマンドをホスト上で直接実行した場合とは異なる情報を特権コンテナー内で確認することができます。情報が異なるのは、2 つの環境がそれぞれ独自のマウントテーブルを保持しているためです。

10.3.3. プロセステーブル

コンテナー内で実行しているプロセスのみを表示する通常のコンテナーとは異なり、特権コンテナー内で (**--pid=host** を設定して) **ps -e** コマンド を実行することにより、ホスト上で実行するすべてのプロセスを表示できます。したがって、ホストから特権コンテナーで実行したコマンドにプロセス ID を渡すことができます (例: **kill PID**)。ただし一部のコマンドでは、コンテナーからプロセスにアクセスしようとする際にアクセス権の問題が発生する場合があります。

10.3.4. プロセス間通信

ホストの IPC 機能は、特権コンテナーからアクセスできます。そのため、**ipcs** などのコマンドを実行すると、ホストでアクティブなメッセージキュー、共有メモリーセグメント、およびセマフォセットの情報を確認できます。

第11章 ATOMIC TOOLS CONTAINER イメージの使用

11.1. 概要

Red Hat Enterprise Linux Atomic Tools Container (RHEL Tools Container) は、Red Hat Enterprise Linux Atomic (RHEL Atomic) Host のトラブルシューティングおよび調査を行うための数百のソフトウェアツールが含まれる Docker フォーマットのイメージです。RHEL Tools Container は特権コンテナーとして実行するように設計されており、これを使用すると、RHEL Atomic Host システムと直接対話して問題を明らかにし、解決することができます。RHEL Tools Container には、使用頻度の高い sosreport、kdump などのツール (その大半は RHEL Atomic に含まれていない) があります。

本書では、以下を説明します。

- RHEL Tools Container を取得し、実行する方法
- RHEL Tools Container の機能のしくみ
- RHEL Tools Container で使用できるコマンドとそれらの使用方法

11.2. RHEL TOOLS CONTAINER の概要

RHEL Atomic は、軽量な簡易バージョンの Red Hat Enterprise Linux であり、Linux コンテナーを実行するために設定され、調整されました。軽量にすることで、デプロイ時やデプロイ後に効率的に実行できるように、消費するリソース量を最小限に抑えるようにしています。したがって RHEL Atomic は、とくにクラウド環境でコンテナーをホストするのに適しています。

Atomic のサイズを小さくしているのは、標準の RHEL システムで利用できるツールの多くが Atomic にはインストールされていないためです。さらに、Atomic には追加のソフトウェアパッケージがインストールできないようになっています (**yum install favorite_RPM** はサポートされていません)。

この問題は、RHEL Tools Container を RHEL Atomic システムに導入すれば解決できます。この方法は、Atomic システムを最初にデプロイした場合や、問題が発生し、トラブルシューティングに使用する追加ツールが必要になった場合に利用できます。

以下は、RHEL Tools Container に関するいくつかの注意点です。

- **サイズが大きい**: コンテナーなので、サイズが非常に大きくなります (現時点では約 1GB)。これは、コンテナーに Atomic の監視やトラブルシューティングに必要なツールをできるだけ組み込む必要があるためです。通常の操作で領域の消費が問題となる場合は、必要に応じてコンテナーを Atomic システムに配置できます (ただしコンテナーを取得する場合、問題を急いで修正したい場合でもかなりの時間がかかる可能性があることに注意してください)。
- **man ページが組み込まれている**: このコンテナーは、コンテナーで RHEL ドキュメントを利用する方法を提供します。man コマンドは RHEL Atomic には組み込まれていないため、RHEL Tools Container から man ページを表示することができます。さらに /usr/share/doc のすべてのコンテンツは、そのコンテナーにインストールされたすべてのパッケージ用に組み込まれます。
- **特権を付与する**: デフォルトで、コンテナーは Atomic ホストのファイルシステムまたは名前空間 (ネットワーキング、IPC、プロセステーブルなど) のほとんどを表示することはできません。しかし、RHEL Tools Container は特権付きのホストとして実行され、ホストの名前空間と機能へのアクセスを開くため、そのコンテナーから実行するほとんどのコマンドは、ホストで直接実行されているかのように、ホストで表示して動作できます。

- **動作が異なる場合がある**: コマンドをこのコンテナー内から実行すると、特権を付与した場合でも、RHEL ホストシステムから直接実行した時と動作が異なる場合があります。本書では、RHEL Tools Container に含まれる最も便利なツールの一部を説明しており、特権付きのコンテナーで実行すると、コマンドの予想される動作がどのように異なるかについて説明します。

11.3. RHEL TOOLS CONTAINER の取得および実行

RHEL Tools Container は RHEL Atomic Host で実行されるように設計されています。そのため、これを使う前に RHEL Atomic システムをインストールする必要があります。次に、以下の手順に従って RHEL Tools Container の取得し、ロードし、実行します。

- **RHEL Atomic Host のインストール**: RHEL Atomic Host をインストールし、設定するには、ドキュメンテーションのページに一覧表示されているインストールガイドを参照してください。
- **RHEL Tools イメージの取得**: RHEL Atomic Host にログインしたら、以下のように **docker pull** コマンドを実行して RHEL Tools Container を取得します。

```
# docker pull rhel7/rhel-tools
```

- **RHEL Tools Container の起動**: RHEL Tools Container を実行するには、atomic コマンドを使用します。以下のコマンドを実行すると、適切なオプションと共に docker コマンドを使用してコンテナーを起動できます。

```
# atomic run rhel7/rhel-tools
[root@localhost /]#
```

これでシェルがコンテナー内に開き、そのコンテナー内のすべてのツールを実行できるようになりました。準備ができたら exit を実行します。次のセクションでは、RHEL Tools Container から実行する可能性のあるいくつかのコマンド例を示します。

11.4. RHEL TOOLS CONTAINER からのコマンドの実行

以下のセクションでは、RHEL Tools Container で利用可能なコマンドの説明と、コンテナーの内外ではコマンドの動作がどのように異なるかについて説明します。

- **blktrace**: **blktrace** をコンテナー内で使用するには、まず debugfs ファイルシステムをマウントする必要があります。以下は、そのファイルシステムをマウントし、blktrace を実行する例です。

```
# mount -t debugfs debugfs /sys/kernel/debug/
# blktrace /dev/vda
^C
=== vda ===
  CPU  0:                38086 events,     1786 KiB data
  Total:                 38086 events (dropped 0),     1786 KiB
data
```

- **sosreport**: **sosreport** コマンドには、コンテナーを認識するための atomic プラグインが含まれます。そのため、**sosreport** を単純に実行し、ホストで直接実行した場合とほぼ同じ結果が得られます。コンテナー内から以下を実行できます。

```
# sosreport
Please enter your first initial and last name
[localhost.localdomain]: jjones
```

```
Please enter the case id that you are generating this report for:
12345678
...
# ls /host/var/tmp
sosreport-jjones.12345678-20150203102944.tar.xz
sosreport-jjones.12345678-20150203102944.tar.xz.md5
```

レポートはホストの **/var/tmp** ディレクトリーにコピーされることに留意してください。ホストの root ファイルシステムはコンテナー内の **/** にマウントされるため、レポートはコンテナーの **/host/var/tmp** ディレクトリーで利用できます。そのため、レポートはコンテナーを閉じた後も利用できます。

- **useradd**: root 以外のアクティビティーを実行するユーザーをコンテナーに追加する場合は、**useradd** コマンドを使用してから、追加手順に従ってホームディレクトリーを作成します。

```
# useradd jjones
# su - jjones
[jjones@example ~]$
```

- **strace**: ホストのプロセステーブルは RHEL Tools Container から表示できるため、プロセス ID を引数として使用する多くのコマンドはコンテナー内から動作します。以下は、strace コマンドの例です。

```
# ps -ef | grep ssh
root        998      1  0 Jan29 ?        00:00:00 /usr/sbin/sshd -D
# strace -p 998
Process 998 attached
select(7, [3 4], NULL, NULL, NULL ...
```

11.5. RHEL TOOLS CONTAINER を実行するためのヒント

以下は、RHEL Tools Container の実行に関連するその他の注意点です。

- コンテナーを明示的に削除しない場合 (**docker rm rhel-tools**)、コンテナーは引き続きシステムに残ります。
- コンテナーは削除されずに存在し続けるため、実行するすべての変更 (例: **yum install package** を実行) は、コンテナーを実行するたびに永続化します。そのため、**atomic run rhel7/rhel-tools** を実行してもファイルは取得されず、2 回目に実行する場合でもホストで追加のセットアップは実行されません。
- イメージは **rhel7/rhel-tools** という名前で特定されますが、イメージが実行されると、実行中のインスタンスは **rhel-tools** という名前のコンテナーとして参照されます。コンテナーはデフォルトで削除されないため、コンテナーの名前は、停止した後でも **docker ps -a** を実行して確認できます。
- **rhel-tools** コンテナーは削除後も保持されるため、古いバージョンを明示的に削除しないと新規バージョンのコンテナーにアップグレードすることはできません。これを実行するには、維持する必要のあるコンテナーのファイルを保存する必要があります (それらを **/host** の任意の場所にコピーする)。次に **docker rm rhel-tools** と入力します。その後に **docker pull rhel7/rhel-tools** を新たに実行します。
- Atomic ホストで直接実行する必要のあるコマンドには、systemd 関連 (**systemctl** および **journalctl**)、LVM (**lvm**、**lvdisplay**、**vgdisplay** など)、**atomic** コマンド、およびブロックデバイスを変更するすべてのコマンドが含まれます。

- **subscription-manager** コマンドは、RHEL Atomic Host および RHEL Tools Container 内の両方で使用できます。Atomic では、有効な Red Hat サブスクリプションをホストに割り当てる必要があります。コンテナーには、関連する man ページを利用可能にする **subscription-manager** パッケージがあります。**subscription-manager** コマンドはコンテナー内で実行することはできませんが、ホストのサブスクライブを行うと、コンテナー内で **yum** コマンドを使用してコンテナーにソフトウェアパッケージを追加したり、管理したりすることができます。
- RHEL Tools Container に関連する問題がある場合は、**bugzilla.redhat.com** にバグを提出し、RFE を作成することができます。「Red Hat Enterprise Linux」製品の「rhel-tools-docker」コンポーネントを選択してください。

第12章 ATOMIC RSYSLOG コンテナーイメージの使用

12.1. 概要

Red Hat Enterprise Linux rsyslog Atomic コンテナーイメージは、Red Hat Enterprise Linux Atomic (RHEL Atomic) Host で実行するために設計された Docker フォーマットのイメージです。

このコンテナーを使って、以下を実行する rsyslogd デーモンを起動できます。

- Atomic Host のファイルシステムに保存される設定ファイルおよびログファイルを使用する。
- ログメッセージをリモートログホストに送る機能を含む、標準の rsyslog 機能を提供するように設定する。

本書では、RHEL rsyslog コンテナーの取得および実行方法について説明します。

rsyslog サービスは Red Hat Enterprise Linux Atomic Host にインストールされていないため、rsyslog コンテナーが、そのサービスを Atomic Host に追加する方法を提供します。

以下は、rsyslog コンテナーのいくつかの機能です。

- **atomic コマンドからのインストール**: **atomic install** コマンドを使用して rsyslog コンテナーを取得して実行すると、いくつかのことが生じます。コンテナー自体をレジストリーから取得し、rsyslog サービスが必要とするファイルおよびディレクトリーはホストに追加され、コンテナーは **docker run** で起動します。
- **ホストからの設定**: rsyslog サービスで必要なファイルは Atomic Host に保存されるため、コンテナーの中に移す必要はありません。すべての設定はホストから実行できます。
- **サービスの再起動**: 設定に変更した場合、変更を取得するには、コンテナーを停止し、削除し、再起動する必要があります (**docker stop rsyslog; docker rm rsyslog; atomic run rhel7/rsyslog**)。
- **スーパー特権コンテナー**: rsyslog コンテナーを実行すると、そのコンテナーからホストシステムへの特権が付与されることに留意してください。コンテナーには RHEL Atomic Host への root アクセスがあり、特権付きの設定およびログファイルへのアクセスが行われます。

12.2. RHEL RSYSLOG コンテナーの取得および実行

RHEL Atomic Host で rsyslog Atomic コンテナーイメージを使用するには、以下の手順に従ってイメージのインストール、ロードを行ってから実行する必要があります。

1. **RHEL Atomic Host のインストール**: RHEL Atomic Host をインストールし、設定するには、Red Hat Enterprise Linux Atomic Host ドキュメンテーション のページに一覧表示されているインストールガイドを参照してください。
2. **RHEL rsyslog コンテナーのインストール**: RHEL Atomic Host にログインしたら、以下のコマンドを実行して RHEL rsyslog コンテナーを取得し、これを起動します。

```
# docker pull rhel7/rsyslog
# atomic install rhel7/rsyslog
...
docker run --rm --privileged -v /:/host -e HOST=/host -e
```

```
IMAGE=rhel7/rsyslog -e NAME=rsyslog rhel7/rsyslog /bin/install.sh
Creating directory at /host//etc/pki/rsyslog
Installing file at /host//etc/rsyslog.conf
Installing file at /host//etc/sysconfig/rsyslog
```

3. **rsyslog コンテナーの起動**: RHEL rsyslog コンテナーを実行するには、atomic コマンドを使用します。以下のコマンドを実行すると、適切なオプションと共に docker コマンドを使用してコンテナーを起動できます。

```
# atomic run rhel7/rsyslog
docker run -d --privileged --name rsyslog --net=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v
/etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log -v
/var/lib/rsyslog:/var/lib/rsyslog -v /run/log:/run/log -v
/etc/machine-id:/etc/machine-id -v /etc/localtime:/etc/localtime
-e IMAGE=rhel7/rsyslog -e NAME=rsyslog --restart=always
rhel7/rsyslog /bin/rsyslog.sh
5803dbade82274158f0694a19fdcd7aac044a2656b2ce96d1aebdb0e30ad5ffd
```

atomic コマンドを起動したら、ryslog コンテナーを起動するために実行する 'docker' コマンドを確認できます。rsyslogd コンテナーは、スーパー特権コンテナーとして実行されます。

4. **コンテナーが実行中であることの確認**: 以下を入力して rsyslog コンテナーが実行中であることを確認します。

```
# docker ps
CONTAINER ID IMAGE
COMMAND             CREATED       STATUS        PORTS NAMES
5803dbade822 registry.access.stage.redhat.com/rhel7/rsyslog:7.1-3
"/bin/rsyslog.sh"   9 minutes ago Up 9 minutes        rsyslog
```

注記

"registry.access.redhat.com/rhel7/rsyslog:7.1-3" は、ダウンロードしたレジストリーの名前と、取得したイメージのバージョンの両方が含まれたイメージの正式な名前です。コンテナー自体はローカルで実行されますが、単に rsyslog と呼ばれます。イメージとコンテナーでは、docker の動作方法が異なります。

5. **rsyslog サービスが機能していることの確認**: メッセージが **/var/log/messages** ファイルに保存されたら、シェルで以下を入力します。

```
# tail -f /var/log/messages
```

6. **ログメッセージの生成**: 以下を入力してログメッセージを生成します。

```
# logger "Test that rsyslog is doing great"
```

rsyslog サービスが機能している場合は、メッセージが tail コマンドを実行しているシェルから表示されるはずです。Atomic Host で rsyslog サービスを使用できるようになりました。

12.3. RSYSLOG コンテナーを実行するためのヒント

以下は、RHEL rsyslog コンテナーの実行に関連するその他のいくつかの注意点です。

- **永続的なログについて**: デフォルトでは、Red Hat Enterprise Linux Atomic Host システムは、**/etc/systemd/journald.conf** で以下の値を設定し、journald を使用してローカルの root ファイルシステムで永続ログにログを記録します。

  ```
  Storage=persistent
  ```

  永続ログをローカルの rsyslog ログか、またはリモートの rsyslog サーバーのいずれかに設定するには、その行を以下のように変更することで、ローカルの journald 永続ログを無効にする必要がある場合があります。

  ```
  Storage=volatile
  ```

  さらに、RHEL Atomic Host システムを再起動します。この場合、journald は ramdisk にローカルログを維持しますが、ディスクには書き込みません。これにより、データが別の場所に安全に取得されている場合は、ローカルディスク IO で保存されます。rsyslog コンテナーは依然として journald ログを取得し、これを処理できます。

- **rsyslog 設定の変更**: rsyslog コンテナーの設定を変更するたびに、実行中の rsyslog コンテナーを停止し、削除してから、新規のコンテナーを起動する必要があります。これを実行するには、以下のコマンドを実行します。

  ```
  # docker stop rsyslog
  # docker rm rsyslog
  # atomic run rhel7/rsyslog
  ```

- **ログローテーション**: rsyslog コンテナーイメージの初期バージョンでは、rsyslog ログファイルのローカルローテーションをサポートしません。このサポートは将来のアップデートで追加されます。ただし領域に余裕がある場合は、rsyslog をローカルのログファイルで使用できます。

  rsyslog がリモートのログ収集ホストにのみログを送信するよう設定されている場合は、ローカルのログローテーションに 要件はありません。rsyslog でローカルおよびリモートのログ記録を設定する方法の詳細は、Red Hat Enterprise Linux システム管理者のガイド を参照してください。

- **ログを取得するのに十分な領域があることを確認する**

  - 数多くのクラウド環境に理想的な設定は、リモートの rsyslog サーバーにログを記録するように rsyslog を設定する方法です。
  - ローカルストレージにログを記録する場合は、コンテナー内でログローテーションが発生していないことを確認します。今後のリリースでは、logrotate 設定ファイル (**/etc/logrotate.d/** ディレクトリーのファイルや **/etc/logrotate.conf** ファイルなど) を編集して、rsyslog 設定にログファイルのサイズ制限を設定することをサポートします。この機能はまだサポートされていません。
  - とくに atomic host qcow2 イメージの root ファイルシステムで利用できる領域の大きさが制限されていることに注意してください。より大きな領域のプロビジョニングは、Red Hat Enterprise Linux Atomic Host の anaconda インストーラー ISO イメージを使ってインストールすることで実行できます。

- **イメージおよびコンテナーのライフサイクル**

  Red Hat Enterprise Linux rsyslog Atomic コンテナーイメージの新規バージョンにアップグレー

ドする必要がある場合、docker pull rhel7/rsyslog* を実行して新規イメージをダウンロードするだけでは不十分です。さらに、新規イメージから新規コンテナーを作成するには、再実行する前に、以下のコマンドを実行して既存の rsyslog コンテナーを明示的に削除する必要があります。

```
# docker pull rhel7/rsyslog  If a new image downloads, run the
following:
# docker stop rsyslog
# docker rm rsyslog
# atomic install rhel7/rsyslog
# atomic run rhel7/rsyslog
```

第13章 ATOMIC SYSTEM ACTIVITY DATA COLLECTOR (SADC) コンテナーイメージの使用

13.1. 概要

Red Hat Enterprise Linux sadc Atomic コンテナーイメージは、sysstat パッケージに含まれるシステム監視やデータ収集ユーティリティーを Docker フォーマットのコンテナーとして提供します。このコンテナーは、Red Hat Enterprise Linux Atomic Host で実行するために設計されています。このコンテナーをインストールして実行すると、Atomic システムで以下が行われます。

- システムアクティビティーのデータが継続的に収集される。
- **cifsiostat**、**iostat**、**mpstat**, **nfsiostat**、**pidstat**、 **sadf**、および **sar** などのコマンドを使用して上記のデータを表示する。このコマンドは **docker exec sadc** コマンドから使用できます。

本書では、sadc コンテナーを取得し、実行する方法について説明します。

13.2. SADC コンテナーの概要

sysstat パッケージ (sar、iostat、sadc その他ツールを含む) は Red Hat Enterprise Linux Atomic Host にインストールされていないため、sadc コンテナーがこれらのユーティリティーを Atomic Host に追加する方法を提供します。以下は、sadc コンテナーの各種機能の一部です。

- **atomic コマンドからのインストール**: "atomic install" コマンドを使用して sadc コンテナーの取得して実行すると、以下が実行されます。まずコンテナー自体がレジストリーから取得され、sadc サービスで必要なファイルおよびディレクトリーはホストに追加され、コンテナーは **docker run** で起動します。
- **ホストからの設定**: sadc データ収集サービスで必要なファイルは Atomic Host に保存されるため、コンテナー内に移す必要はありません。すべての設定はホストから実行できます。
- **スーパー特権コンテナー**: sadc コンテナーを実行すると、そのコンテナーからホストシステムへの特権が付与されることに留意してください。コンテナーには RHEL Atomic Host への root アクセスがあり、特権付きの設定およびログファイルへのアクセスが行われます。特権コンテナーについての詳細は、「Running Privileged Docker Containers in RHEL Atomic (RHEL Atomic での特権 Docker コンテナーの実行)」に関する情報を参照してください。

13.3. RHEL SADC コンテナーを取得および実行する方法

Red Hat Enterprise Linux Atomic Host で sadc コンテナーを使用するには、以下の手順で説明されているように、これをインストールし、ロードし、実行する必要があります。

1. **RHEL Atomic Host のインストール**: RHEL Atomic Host をインストールし、設定するには、Red Hat Enterprise Linux Atomic Host ドキュメンテーションのページに一覧表示されている該当するインストールガイドを参照してください。
2. **RHEL sadc コンテナーのインストール**: RHEL Atomic Host にログインしたら、以下のコマンドを実行して sadc コンテナーを取得し、これを起動します。

```
# docker pull rhel7/sadc
# atomic install rhel7/sadc
docker run --rm --privileged --name sadc -v /:/host -e HOST=/host
-e IMAGE=rhel7/sadc -e NAME=name rhel7/sadc
```

```
/usr/local/bin/sysstat-install.sh
Installing file at /host//etc/cron.d/sysstat
Installing file at /host//etc/sysconfig/sysstat
Installing file at /host//etc/sysconfig/sysstat.ioconf
Installing file at /host//usr/local/bin/sysstat.sh
```

3. **sadc コンテナーの起動**: RHEL sadc コンテナーを実行するには、**atomic** コマンドを使用します。以下のコマンドは、適切なオプションと共に docker コマンドを使用してコンテナーを起動します。

```
# atomic run rhel7/sadc
docker run -d --privileged --name sadc -v
/etc/sysconfig/sysstat:/etc/sysconfig/sysstat -v
/etc/sysconfig/sysstat.ioconf:/etc/sysconfig/sysstat.ioconf -v
/var/log/sa:/var/log/sa -v /:/host -e HOST=/host -e
IMAGE=rhel7/sadc -e NAME=sadc --net=host --restart=always
rhel7/sadc /usr/local/bin/sysstat.sh
11c566e20ec995a164f815d9bb76b4b876c555f507c9f56c41f5009c9b1bebf4
```

atomic コマンドが起動すると、sadc コンテナーを起動するために実行される正確な docker コマンドが表示されます。sadc コンテナーはスーパー特権コンテナーとして実行されます。スーパー特権付きコンテナーについての詳細は、「Running Super Privileged Docker Containers on a Red Hat Enterprise Linux Atomic Host (Red Hat Enterprise Linux Atomic Host でスーパー特権 Docker コンテナーを実行する)」を参照してください。

4. **コンテナーが実行中であることの確認**: 以下を入力して sadc コンテナーが実行中であることを確認します。

```
# docker ps
CONTAINER ID IMAGE
COMMAND              CREATED       STATUS       PORTS NAMES
11c566e20ec9 registry.access.stage.redhat.com/rhel7/sadc:7.1-3
"/usr/local/bin/syss 3 minutes ago Up 2 minutes       sadc
```

注意: "registry.access.redhat.com/rhel7/sadc:7.1-3" は、ダウンロードしたレジストリーの名前と取得したイメージのバージョンの両方を含むイメージの正式な名前です。コンテナー自体はローカルで実行しますが、単に「sadc」と呼ばれます。イメージとコンテナーでは、docker の動作方法が異なります。

5. **sadc データの生成**: シェルに以下を入力して、システムアクティビティーのデータを生成し、sadc が適切に動作していることを確認します。

```
# docker exec sadc /usr/lib64/sa/sa1 1 1
```

6. **sadc が適切に機能することの確認**: sadc がシステムアクティビティーのデータを生成したら、以下のように sar コマンドを使用して確認できます。

```
# docker exec sadc sar
Linux 3.10.0-229.el7.x86_64 (minion1.example.com) 02/27/15
_x86_64_ (1 CPU)

09:31:25          LINUX RESTART
09:32:00 CPU %user  %nice %system  %iowait  %steal   %idle
09:32:18 all 0.86   0.00    0.92      0.00    0.00   98.22
```

sadc が動作している場合、実行したばかりの sadc コマンドで生成されたデータを確認できます。

新規のデータは 10 分ごとに生成されます。そのため、**sar** コマンドを再度実行して、データが継続的に収集されていることを確認します。

13.4. SADC コンテナーを実行するためのヒント

以下は、sadc コンテナーの実行に関連するその他の注意点です。

- **sysstat コマンドの実行**: sysstat パッケージのいずれかのコマンドを実行して、sadc コンテナーで収集されたデータを表示できます。これらには、**cifsiostat**、**iostat**、**mpstat**、**nfsiostat**、**pidstat**、**sadf**、および **sar** が含まれます。これらのコマンドは Atomic Host 上にないため、**docker exec** を使用して実行する必要があります。たとえば、以下のようになります。

  ```
  # docker exec sadc iostat
  ```

- **イメージおよびコンテナーのライフサイクル**

  Red Hat Enterprise Linux sadc Atomic コンテナーイメージの新規バージョンにアップグレードする必要がある場合、**docker pull rhel7/sadc** を実行して新規イメージをダウンロードするだけでは不十分です。さらに、新規イメージから新規コンテナーを作成するには、再実行する前に、以下のコマンドを実行して既存の sadc コンテナーを明示的に削除する必要があります。

  ```
  # docker stop sadc
  # docker rm sadc
  ```