Red Hat Enterprise Linux 8

コンテナーの構築、実行、および管理

Red Hat Enterprise Linux 8 での Linux コンテナーの構築、実行、および管理

Last Updated: 2020-09-07

Red Hat Enterprise Linux 8 コンテナーの構築、実行、および管理

Red Hat Enterprise Linux 8 での Linux コンテナーの構築、実行、および管理

法律上の通知

概要

本ガイドでは、podman、buildah、skopeo、runc などのコマンドラインツールを使用して、RHEL 8 システムで Linux コンテナーを使用する方法を説明します。

目次

はじめに

Red Hat は、コンテナーのユースケースを、シングルノードとマルチノードという別々のグループに分類します。マルチノードは、分散システムと呼ばれることもあります。OpenShift は、コンテナー化されたアプリケーションのパブリックでスケーラブルなデプロイメントを提供するために構築されました。ただし、OpenShift 以外では、コンテナーを操作する、小規模で機敏なツールセットがあると便利です。

参照するコンテナツールセットは、シングルノードのユースケースで使用できます。ただし、既存のビルドシステム (CI/CD 環境) にこのツールを接続することもできます。また、これを使用して、ビッグデータなどのワークロード固有のユースケースを処理することもできます。Red Hat Enterprise Linux (RHEL) 8 は、シングルノードのユースケースを対象として、個々のコンテナーを検索、実行、ビルド、および共有するツールセットを提供します。

本ガイドでは、podman、buildah、skopeo、runc などのコマンドラインツールを使用して、RHEL 8 システムで Linux コンテナーを使用する方法を説明します。以上のツールに加えて、Red Hat は、お客様が所有するイメージの基盤として機能するベースイメージを提供します。このベースイメージの一部は、ビジネスアプリケーション (Node.js、PHP、Java、Python など) からインフラストラクチャー (ロギング、データ収集、認証など) までのユースケースを対象としています。

RED HAT ドキュメントへのフィードバック (英語のみ)

ご意見ご要望をお聞かせください。ドキュメントの改善点はございませんか。改善点を報告する場合は、以下のように行います。

- 特定の文章に簡単なコメントを記入する場合は、以下の手順を行います。
  1. ドキュメントの表示が **Multi-page HTML** 形式になっていて、ドキュメントの右上端に **Feedback** ボタンがあることを確認してください。
  2. マウスカーソルで、コメントを追加する部分を強調表示します。
  3. そのテキストの下に表示される **Add Feedback** ポップアップをクリックします。
  4. 表示される手順に従ってください。
- より詳細なフィードバックを行う場合は、Bugzilla のチケットを作成します。
  1. Bugzilla の Web サイトにアクセスします。
  2. Component で **Documentation** を選択します。
  3. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも記入してください。
  4. **Submit Bug** をクリックします。

第1章 コンテナーの使用

Linux コンテナーは、イメージベースのデプロイメント方法の柔軟性と、軽量アプリケーションの分離を組み合わせた、主要なオープンソースアプリケーションをパッケージ化して、配信するテクノロジーとして登場しました。

Red Hat Enterprise Linux は、リソース管理用のコントロールグループ (Cgroup)、プロセス分離用の名前空間、SELinux によるセキュリティーなどのコアな技術を使用して Linux コンテナーを実装します。これにより、セキュアなマルチテナントが可能になり、セキュリティーの悪用の可能性が低減します。これは、エンタープライズ品質のコンテナーを生成および実行する環境を提供することを目的としています。

Red Hat OpenShiftは、**Pod** と呼ばれる単位でコンテナーを構築、管理、および実行を行う、強力なコマンドラインと Web UI ツールを提供します。ただし、OpenShift 外で、個々のコンテナーおよび コンテナーイメージ を構築および管理したい場合があります。RHEL システムで直接実行するこのようなタスクを実行するツールは、本ガイドで説明します。

他のコンテナツールの実装とは異なり、ここで説明するツールはモノリシック Docker の コンテナーエンジン と、**docker** コマンドを中心としたものではありません。代わりに、コンテナーエンジンがなくても動作できる一連のコマンドラインツールを提供します。これには、以下が含まれます。

- **podman** - Pod ポッドおよびコンテナーイメージを直接管理 (run、stop、start、ps、attach、exec など)
- **buildah** - コンテナーイメージの構築、プッシュ、および署名
- **skopeo** - イメージのコピー、検証、削除、および署名
- **runc** - podman および buildah に、コンテナーの実行機能と構築機能を提供

これは、このツールが、Docker が生成して管理するのと同じ Linux コンテナーや、その他の OCI 互換コンテナー エンジン の管理に使用する Open Container Initiative (OCI) と互換性があるためです。ただし、シングルノードのユースケースでは、Red Hat Enterprise Linux で直接実行することが特に適しています。

マルチノードのコンテナープラットフォームは、OpenShift を参照してください。本書で説明されているシングルノード、デーモンレスツールの代わりに、OpenShift ではデーモンベースのコンテナーエンジンが必要です。詳細は「CRI-O コンテナーエンジンの使用」を参照してください。

1.1. DOCKER を使用せずにコンテナーを実行

Red Hat が OpenShift から削除したのは、Docker コンテナーエンジンだけでははありません。**docker** コマンドと Docker コンテナーエンジンも、Red Hat Enterprise Linux 8 から完全に削除しました。RHEL 8 には Docker が含まれず、Red Hat のサポート対象外になります (ただし、他のソースから引き続き利用できます)。

Docker の削除は、コンテナーがどのように扱われるかに関する Red Hat の考え方の変更を反映しています。

- 企業は、コマンドラインからコンテナーを個別に実行することを重視していません。コンテナーは、主に OpenShift などの、Kubernetes ベースのプラットフォームで使用されます。
- OpenShift を、コンテナーを実行するプロジェクトとして再配置することで、Docker などのコンテナーエンジンが、エンドユーザーによる直接アクセスがない、OpenShift の別のコンポーネントになります。

- OpenShift のコンテナーエンジンは直接使用することを目的としていないため、スタンドアロン機能の多くを実装しなくても、OpenShift が必要とする機能をすべて実行することに焦点をあてる、限定された機能セットで実装できます。

RHEL 8 では Docker が 提供されなくなり、シングルノードの使用には OpenShift のコンテナーエンジンが含まれなくなりましたが、コンテナーとイメージは、引き続きコマンドを使用して手動で操作したいという声がありました。そこで Red Hat は、**docker** コマンドが行う作業のほとんどを実装するツールセットの作成に取り掛かりました。

このような **docker** コマンド機能を引き継ぐために、**podman**、**skopeo**、 **buildah** などのツールが開発されました。このシナリオの各ツールはより軽量になり、機能のサブセットに焦点があたっています。また、このツールでは、コンテナーエンジンを実装するために実行するデーモンプロセスが必要なく、デーモンプロセスを使用するオーバーヘッドを使用せずに実行できます。

RHEL 8 で Docker を使用する場合は、別のアップストリームプロジェクトから Docker を取得できるものの、RHEL 8 ではその使用がサポートされない点に注意してください。**podman** に正確に実装されている **docker** コマンドライン機能が多すぎるため、**docker** と入力して podman を実行するエイリアスを設定できます。

podman-docker パッケージをインストールすると、このようなエイリアスが設定されます。**docker** コマンドラインを実行すると、実際には **podman** が実行します。このパッケージは後で追加してください。

1.2. コンテナーの RHEL アーキテクチャーの選択

Red Hat は、以下のコンピューターアーキテクチャーに、コンテナーイメージとコンテナー関連のソフトウェアを提供します。

- AMD64 および Intel 64 (ベースイメージおよびレイヤー構造イメージ) (32 ビット AMD および Intel アーキテクチャーはサポートされません)
- PowerPC 8 および 9 の 64 ビット (ベースイメージおよび大概のレイヤー構造イメージ)
- IBM Z (ベースイメージと、大概のレイヤー構造イメージ)
- ARM 64 ビット (ベースイメージのみ)

初めは、すべてのアーキテクチャーですべての Red Hat イメージがサポートされたわけではありませんが、一覧に挙げられているすべてのアーキテクチャーでほぼすべてが利用可能になりました。サポートされるイメージの一覧は「Universal Base Images (UBI): Images, repositories, and packages」 を参照してください。

1.3. コンテナーツールの取得

個々のコンテナーを操作できる環境を用意するには、Red Hat Enterprise Linux 8 システムをインストールしてから、コンテナーを検索、実行、構築、および共有する一連のコンテナーツールを追加します。以下は、RHEL 8 でインストールできるコンテナー関連のツールの例です。

- **podman** - コンテナーを管理するクライアントツール。個々のコンテナーおよびイメージを処理する **docker** コマンドのほとんどの機能を置き換えることができます
- **buildah** - OCI 準拠のコンテナーイメージを構築するクライアントツール。
- **skopeo** - コンテナーレジストリーとの間で、コンテナーイメージをコピーするクライアントツール。イメージの署名および認証を行う機能も含まれています。

- **runc** - Open Container Initiative (OCI) 形式のコンテナーを実行および操作するコンテナーランタイムクライアント。

RHEL サブスクリプションモデルを使用してコンテナーイメージを作成する場合は、構築するホストコンピューターを正しく登録し、権利を付与する必要があります。コンテナーを構築するプロセスの中でパッケージをインストールすると、ビルドプロセスが、RHEL ホストで利用可能なエンタイトルメントに自動的にアクセスできるようになります。そのため、そのホストで有効なリポジトリーから RPM パッケージを取得できます。

1. **RHEL のインストール** - 準備が整った場合は、Red Hat Enterprise Linux システムをインストールして開始できます。

2. **RHEL の登録** - RHEL をインストールしたら、システムを登録します。ユーザー名とパスワードを入力するように求められます。ユーザー名とパスワードは、Red Hat カスタマーポータルのログイン認証情報と同じであることに注意してください。

```
# subscription-manager register
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: ********
Password: **********
```

3. **RHEL のサブスクライブ** - 自動的にサブスクライブするか、Red Hat Enterprise Linux を含むサブスクリプションのプール ID を指定します。以下は、サブスクリプションの自動割り当ての例です。

```
# subscription-manager attach --auto
```

4. **パッケージのインストール** - 各コンテナーを構築して作業を開始するには、container-tools モジュールをインストールします。このモジュールは、コンテナーソフトウェアパッケージを完全セットで取得します。

```
# yum module install -y container-tools
```

5. **podman-docker のインストール (任意)** - **docker** コマンド、または **docker** を直接呼び出すスクリプトを使用して、podman-docker パッケージをインストールできます。そのパッケージは、**docker** コマンドラインインターフェースを、一致する **podman** コマンドに置き換えるリンクをインストールします。また、man ページも一緒にリンクします。したがって、**man docker info** は、man ページの **podman info** を表示します。

```
# yum install -y podman-docker
```

1.4. ROOT または ROOT 以外のユーザーとしてコンテナーの実行

スーパーユーザー特権 (root ユーザー) を持つユーザーとして **podman**、**skopeo**、**buildah** などのコンテナーツールを実行すると、コンテナーが、システムで利用できる機能に完全にアクセスできるようにするのが最善の方法です。ただし、RHEL 8.1 以降で一般的に利用可能な「ルートレスコンテナー」と呼ばれる機能を使用すると、コンテナーを一般ユーザーとして使用できます。

Docker などのコンテナーエンジンでは、通常の (root 以外の) ユーザーとして **docker** コマンドを実行できますが、これらの要求を実行する docker デーモンは root として実行されます。そのため、通常のユーザーは、コンテナーを介してシステムに害を及ぼす要求を行うことができますが、要求を行ったユーザーを明確にする必要はありません。ルートレスコンテナーユーザーを設定することにより、システム管理者は、一般ユーザーからのコンテナーアクティビティーに損害を与える可能性を制限しながら、それらのユーザーが自分のアカウントで多くのコンテナー機能を安全に実行できるようにします。

本セクションでは、コンテナーツール (Podman、skopeo、および Buildah) を使用して、root 以外のユーザー (ルートレス) としてコンテナーを操作するように設定する方法を説明します。また、一般ユーザーアカウントは、コンテナーの実行に必要なすべてのオペレーティングシステム機能に完全にアクセスできないため、発生する可能性のある制限についても説明します。

1.4.1. ルートレスコンテナーの設定

root 以外のユーザーでコンテナーツールを使用できるように、RHEL システムを設定する必要があります。

1. **RHEL のインストール** - RHEL 8.1 をインストールするか、RHEL 8.0 から RHEL 8.1 にアップグレードします。この手順に必要な機能は、以前の RHEL 7 バージョンにはありません。RHEL 7.6 以前のバージョンからアップグレードする場合は、この手順を行った後に「ルートレスコンテナーへのアップグレード」に進みます。

2. **podman および slirp4netns のインストール** - podman パッケージおよび slirp4netns パッケージがインストールされていない場合はインストールします。

```
# yum install slirp4netns podman -y
```

3. **ユーザーの名前空間を増やす** - カーネルのユーザー名前空間の数を増やすには、次のコマンドを実行します。

```
# echo "user.max_user_namespaces=28633" > /etc/sysctl.d/userns.conf
# sysctl -p /etc/sysctl.d/userns.conf
```

4. **新規のユーザーアカウントの作成** - 新規のユーザーアカウントを作成し、そのアカウント (例: joe) のパスワードを追加するには、次のコマンドを実行します。

```
# useradd -c "Joe Jones" joe
# passwd joe
```

   ユーザーは、ルートレスな podman を使用できるように自動的に設定されます。

5. **podman コマンドを試用** - 設定したばかりのユーザーとして直接ログインし (正しい環境変数が設定されていないため、**su** または **su -** を使用しないでください)、イメージをプルして実行します。

```
$ podman pull registry.access.redhat.com/ubi8/ubi
$ podman run registry.access.redhat.com/ubi8/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.1 (Ootpa)"
...
```

6. **ルートレス設定の確認** - ルートレス設定が正しく設定されていることを確認するには、**podman unshare** コマンドで変更したユーザー名前空間内でコマンドを実行できます。ルートレスユーザーとして次のコマンドを実行すると、uid がユーザーの名前空間にどのように を割り当てられるかを確認できます。

```
$ podman unshare cat /proc/self/uid_map
         0       1001          1
         1      65537      65536
```

1.4.2. ルートレスコンテナーへのアップグレード

RHEL 7 からアップグレードした場合は、ルートレス Podman を使用できる既存のユーザーに対して、subuid 値および subgid 値を手動で設定する必要があります。

既存のユーザー名およびグループ名 (jill など) を使用して、コンテナーに使用できるアクセス可能なユーザー ID とグループ ID の範囲を設定します。以下にいくつかの警告を示します。

- この範囲には、ルート以外のユーザーの UID と GID を追加しないでください。
- 複数のルートレスコンテナーユーザーを設定する場合は、ユーザーごとに一意の範囲を使用します。
- 既存のコンテナーイメージとの互換性を最大限にするために、UID および GID の数を 65536 とすることが推奨されますが、この数を減らすことができます
- 1000 未満の UID または GID を使用したり、既存のユーザーアカウントから UID または GID を再利用したりしないでください (デフォルトでは 1000 から開始します)。
  以下に例を示します。

  ```
  # echo "jill:165537:65536" >> /etc/subuid
  # echo "jill:165537:65536" >> /etc/subgid
  ```

  これにより、ユーザーまたはグループの jill は、165537~231072 の範囲から、65535 のユーザー ID とグループ ID が割り当てられます。そのユーザーは、コンテナーを操作するコマンドの実行を開始できるはずです。

1.4.3. ルートレスに関する特別な考慮事項

コンテナーを root 以外のユーザーとして実行する場合に考慮すべき事項を次に示します。

- 非 root コンテナーユーザーとして、コンテナーイメージは、**/var/lib/containers** ではなく、ホームディレクトリー (**$HOME/.local/share/containers/storage/**) の下に保存されます。
- ルートレスコンテナーを実行するユーザーには、ホストシステムでユーザー ID およびグループ ID の範囲として実行する特別な権限が付与されます。ただし、これを行わないと、ホストのオペレーティングシステムに root 権限がありません。
- ルートレスコンテナー環境を設定する必要がある場合は、ホームディレクトリー内の設定ファイル (**$HOME/.config/containers**) を編集します。設定ファイルには、**storage.conf** (ストレージ設定用) および **libpod.conf** (さまざまなコンテナー設定用) が含まれます。また、**registries.conf** ファイルを作成し、**podman**を使用してイメージをプル、検索、または実行する時に利用可能なコンテナーレジストリーを識別することもできます。
- ルートレスアカウントで root として実行しているコンテナーは、独自の名前空間内で特権機能を有効にできます。ただし、ホスト上で保護された機能にアクセスするための特別な特権は提供されません (追加の UID および GID が必要です)。以下は、有効ではないルートレスアカウントから作業する可能性があるコンテナーアクションの例です。
  - ホストからマウントされたディレクトリーからアクセスするすべての内容には、コンテナーを実行している UID、またはそのコンポーネントへのアクセス要求にアクセスできない必要があります。
  - 特権なしでは変更できないシステム機能もいくつかあります。たとえば、コンテナー内で SYS_TIME 機能を設定し、ネットワークタイムサービス (ntpd) を実行するだけでは、システムクロックを変更できません。次のような機能を有効にするためには、そのコンテナー

を root として実行し、ルートレスコンテナー環境は回避して root ユーザーの環境を使用する必要があります。

```
$ sudo podman run -d --cap-add SYS_TIME ntpd
```

この例では、ntpd がコンテナー内だけでなく、システム全体の時間の調整を行うことができることに注意してください。

- ルートレスコンテナーは、1024 未満のポートにアクセスすることができません。たとえば、コンテナーの httpd サービスからポート 80 を公開するサービスを開始しますが、名前空間外からはアクセスできません。

```
$ podman run -d httpd
```

ただし、そのポートをホストシステムに公開するには、コンテナーには root ユーザーのコンテナー環境を使用するルート権限が必要です。

```
$ sudo podman run -d -p 80:80 httpd
```

- ワークステーションの管理者は、ユーザーが 1024 未満のサービスを公開できるように構成できますが、セキュリティーへの影響を理解する必要があります。たとえば、一般ユーザーは、公式のポート 80 で Web サーバーを実行し、外部ユーザーが管理者により構成されていると信じ込ませることができます。通常、これはワークステーションでは問題ありませんが、ネットワークにアクセス可能な開発サーバーでは実行できない可能性があり、実稼働サーバーでは実行しないでください。ユーザーがポート 80 にバインドできるようにするには、次のコマンドを実行します。

```
# echo 80 > /proc/sys/net/ipv4/ip_unprivileged_port_start
```

- 現在、ルートレスコンテナーには、静的な subuid と subgid の範囲の設定に依存しています。ユーザー認証を提供するために LDAP または Active Directory を使用している場合は、その UID および GID の範囲をユーザーに提供する自動化された方法はありません。現在の回避策は、/etc/subuid ファイルおよび /etc/subgid ファイルに静的範囲を設定して、使用中の既知の UID および GID と一致させることです。

- リモートファイルシステムは権限のないユーザーの名前空間ではうまく機能しないため、コンテナストレージはローカルファイルシステム上にある必要があります。

- root 権限がない状態で **podman** および関連ツールの実行することの短所は、「Shortcomings of Rootless Podman」を参照してください。

第2章 コンテナーイメージの使用

podman を使用して、コンテナーイメージを実行、起動、停止、調査、および削除できます。

2.1. コンテナーイメージの検索

podman search コマンドを使用すると、イメージ用に選択したコンテナーレジストリーを検索できます。

注記

また、Red Hat Container レジストリーでイメージを検索することもできます。Red Hat Container レジストリーには、イメージの説明、コンテンツ、ヘルスインデックス、その他の情報が含まれます。

設定ファイル **registries.conf** でレジストリーの一覧を確認できます。

```
[registries.search]
registries = ['registry.access.redhat.com', 'registry.redhat.io', 'docker.io']

[registries.insecure]
registries = []

[registries.block]
registries = []
```

- デフォルトでは、**podman search** コマンドは、指定の順序で **[registries.search]** セクションに記載されているレジストリーからコンテナーイメージを検索します。この場合、**podman search** コマンドは、この順番で registry.access.redhat.com、registry.redhat.io、および docker.io で要求されたイメージを検索します。
- **[registries.insecure]** セクションは、TLS を使用しないレジストリーを追加します（非セキュアなレジストリー）。
- **[registries.block]** セクションは、ローカルシステムからレジストリーへのアクセスを許可しません。

root ユーザーとして、**/etc/containers/registries.conf** ファイルを編集し、デフォルトのシステム全体の検索設定を変更できます。

podman の一般 (root 以外の) ユーザーは、ホームディレクトリー (**$HOME/.config/containers/registries.conf**) に独自の **registries.conf** ファイルを作成して、システム全体の設定を上書きできます。

コンテナーレジストリーを設定する際に、以下の条件を満たしていることを確認します。

- 各レジストリーは一重引用符で囲む必要があります。
- **registries = value** に複数のレジストリーが設定されている場合は、それらのレジストリーをコンマで区切る必要があります。
- IP アドレスまたはホスト名のいずれかでレジストリーを識別できます。

- レジストリーが標準以外のポート (安全な場合は TCP ポート 443 で、安全ではない場合は 80) を使用する場合は、ポート番号をレジストリー名とともに入力してください。たとえば、host.example.com:9999 です。

- システムは、**registries.conf** ファイルの **registries.search** リストに表示される順序でレジストリーを検索します。

podman search コマンドの例を以下に示します。最初の例は、quay.io からすべてのイメージの検索に失敗したことを示しています。末尾のスラッシュは、レジストリー全体からアクセス可能なすべてのイメージを検索することを示しています。

```
# podman search quay.io/
ERRO[0000] error searching registry "quay.io": couldn't search registry "quay.io":
unable to retrieve auth token: invalid username/password
```

quay.io レジストリーを検索するには、まずログインします。

```
# podman login quay.io
Username: johndoe
Password: ***********
Login Succeeded!
# podman search quay.io/
INDEX     NAME                              DESCRIPTION   STARS   OFFICIAL   AUTOMATED
quay.io   quay.io/test/myquay                             0
quay.io   quay.io/test/redistest                         0
quay.io   quay.io/johndoe/websrv21                        0
quay.io   quay.io/johndoe/mydbtest                       0
quay.io   quay.io/johndoe/newbuild-10                     0
```

利用可能なすべてのレジストリーで **postgresql** イメージを検索します (40 を超えるイメージが見つかります)。

```
# podman search postgresql-10
INDEX       NAME                                    DESCRIPTION                 STARS OFFICIAL
AUTOMATED
redhat.io   registry.redhat.io/rhel8/postgresql-10          This container image ...       0
redhat.io   registry.redhat.io/rhscl/postgresql-10-rhel7    PostgreSQL is an advanced ...  0
quay.io     quay.io/mettle/postgresql-database-provisioning
docker.io   docker.io/centos/postgresql-10-centos7          PostgreSQL is an advanced ... 13
...
```

postgresql の検索を registry.redhat.io のイメージに制限するには、次のコマンドを入力します。レジストリーおよびイメージ名を入力して、レジストリーの任意のリポジトリーが一致している点に注意してください。

```
# podman search registry.redhat.io/postgresql-10
INDEX       NAME                                    DESCRIPTION          STARS   OFFICIAL
AUTOMATED
redhat.io   registry.redhat.io/rhel8/postgresql-10         This container image ...  0
redhat.io   registry.redhat.io/rhscl/postgresql-10-rhel7   PostgreSQL is an  ...     0
```

各コンテナーイメージについてより詳細な説明を取得するには、**--no-trunc** をコマンドに追加します。

```
# podman search --no-trunc registry.redhat.io/rhel8/postgresql-10
```

```
INDEX      NAME   DESCRIPTION              STARS   OFFICIAL   AUTOMATED
redhat.io  registry.redhat.io/rhel8/postgresql-10
             This container image provides a containerized
             packaging of the PostgreSQL postgres daemon and
             client application. The postgres server daemon
             accepts connections from clients and provides
             access to content from PostgreSQL databases on
             behalf of the clients.   0
```

非セキュアなレジストリーにアクセスするには、レジストリーの完全修飾名を **/etc/containers/registries.conf** ファイルの **[registries.insecure]** セクションに追加します。以下に例を示します。

```
[registries.search]
registries = ['myregistry.example.com']

[registries.insecure]
registries = ['myregistry.example.com']
```

次に、**myimage** イメージを検索します。

```
# podman search myregistry.example.com/myimage
INDEX      NAME  DESCRIPTION                                STARS  OFFICIAL  AUTOMATED
example.com  myregistry.example.com/myimage
  The myimage container executes the ...   0
```

これで、**myimage** イメージをプルできます。

```
# podman pull myimage.example.com/myimage
```

2.2. レジストリーからのイメージの取得 (プル)

リモートレジストリー (Red Hat の独自のコンテナーレジストリーなど) からコンテナーイメージを取得して、ローカルシステムに追加するには、**podman pull** コマンドを使用します。

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

<registry> は、TCP の **<port>** で、コンテナーレジストリーサービスを提供するホストです。**<namespace>** および **<name>** はともに、レジストリーで **<namespace>** で制御される特定のイメージを識別します。<tag> はローカル保存のイメージに対する追加名で、デフォルトのタグは **latest** です。レジストリー、名前空間、イメージ名、タグなどの完全修飾イメージ名を常に使用します。短縮名を使用する場合は、なりすましのリスクが常にあります。不明なユーザーや匿名ユーザーが任意の名前でアカウントを作成できないように、信頼できるレジストリーだけを追加します。

一部のレジストリーは、生の **<name>** をサポートします。ここでは、**<namespace>** は任意です。ただし、それが含まれている場合は、追加の階層レベル **<namespace>** Provides が、同じ **<name>** を持つ複数のイメージを区別するため便利です。以下に例を示します。

名前空間	(<namespace>/<name>) の例
organization	**redhat/kubernetes**、**google/kubernetes**

名前空間	(<namespace>/<name>) の例
login (ユーザー名)	**alice/application**、**bob/application**
role	**devel/database**、**test/database**、**prod/database**

Red Hat が提供するレジストリーは、registry.redhat.io (認証が必要)、registry.access.redhat.com (認証なし)、および registry.connect.redhat.com (Red Hat Partner Connect プログラムイメージを含む) です。registry.redhat.io への移行の詳細は、「Red Hat Container Registry Authentication」 を参照してください。registry.redhat.io からコンテナーを取得する前に、認証する必要があります。以下に例を示します。

```
# podman login registry.redhat.io
Username: myusername
Password: ************
Login Succeeded!
```

pull オプションを使用して、リモートレジストリーからイメージを取得します。rhel ベースイメージおよび rsyslog ロギングイメージを Red Hat レジストリーから取得するには、以下を入力します。

```
# podman pull registry.redhat.io/ubi8/ubi
# podman pull registry.redhat.io/rhel8/rsyslog
```

イメージは、レジストリー名 (registry.redhat.io)、名前空間名 (ubi8)、およびイメージ名 (ubi) で識別されます。タグを追加することもできます (タグを入力しないと、デフォルトの :latest に設定されます)。リポジトリー名 **ubi** は、その前にレジストリーの名前を付けずに **podman pull** コマンドに渡すとあいまいになり、信頼できないレジストリーから生成されたイメージを取得する可能性があります。同じイメージのバージョンが複数ある場合は、**latest** などのタグを追加して、名前を **ubi8/ubi:latest** のようにして、より明示的にイメージを選択します。

上記の **podman pull** コマンドの出力でイメージと、お使いのシステムにあるその他のイメージを一緒に表示するには、**podman images** と入力します。

```
REPOSITORY                      TAG    IMAGE ID      CREATED     SIZE
registry.redhat.io/ubi8/ubi       latest eb205f07ce7d  2 weeks ago 214MB
registry.redhat.io/rhel8/rsyslog  latest 85cfba5cd49c  2 weeks ago 234MB
```

ubi イメージおよび **rsyslog** イメージは、ローカルシステムで利用できるようになりました。

2.3. イメージの調査

podman images を使用すると、ローカルシステムにどのイメージを取得したかを確認できます。イメージに関連するメタデータを表示するには **podman inspect** を使用します。

2.3.1. イメージの一覧表示

ローカルシステムに取得した、利用可能イメージを確認するには、以下のコマンドを実行します。

```
# podman images
REPOSITORY                      TAG      IMAGE ID     CREATED      VIRTUAL SIZE
```

```
registry.redhat.io/rhel8/support-tools   latest   b3d6ce4e0043 2 days ago   234MB
registry.redhat.io/ubi8/ubi-init         latest   779a05997856 2 days ago   225MB
registry.redhat.io/ubi8/ubi              latest   a80dad1c1953 3 days ago   210MB
```

2.3.2. ローカルイメージの検証

ローカルシステムにイメージを取得した後、イメージを実行する前に、そのイメージを調べることが推奨されます。イメージを実行する前に調査する理由としては、以下が挙げられます。

- イメージの役割の理解
- イメージに含まれるソフトウェアの確認

podman inspect コマンドは、イメージの動作に関する基本的な情報を表示します。また、ホストシステムにイメージをマウントするオプションや、ホストからのツールを使用してイメージの内容を調べるオプションもあります。以下は、コンテナーイメージを実行する前に、そのコンテナーを調べる例です。

1. **イメージの検証** - **podman inspect** を実行して、コンテナーイメージを実行する際に実行したコマンドや、その他の情報を確認します。以下は、ubi8/ubi および rhel8/rsyslog のコンテナーイメージを調べる例です (一部の情報だけを抜粋しました)。

```
# podman pull registry.redhat.io/ubi8/ubi
# podman inspect registry.redhat.io/ubi8/ubi | less
...
  "Cmd": [
      "/bin/bash"
  ],
  "Labels": {
      "License": "GPLv3",
      "architecture": "x86_64",
      "authoritative-source-url": "registry.redhat.io",
      "build-date": "2018-10-24T16:46:08.916139",
      "com.redhat.build-host": "cpt-0009.osbs.prod.upshift.rdu2.redhat.com",
      "com.redhat.component": "rhel-server-container",
      "description": "The Red Hat Enterprise Linux Base image is designed to be a fully
supported...
...
```

```
# podman pull registry.redhat.io/rhel8/rsyslog
# podman inspect registry.redhat.io/rhel8/rsyslog
  "Cmd": [
    "/bin/rsyslog.sh"
  ],
  "Labels": {
    "License": "GPLv3",
    "architecture": "x86_64",
...
    "install": "podman run --rm --privileged -v /:/host -e HOST=/host -e IMAGE=IMAGE -e
NAME=NAME IMAGE /bin/install.sh",
...
    "run": "podman run -d --privileged --name NAME --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
```

```
/etc/localtime:/etc/localtime -e IMAGE=IMAGE -e NAME=NAME --restart=always IMAGE
/bin/rsyslog.sh",
    "summary": "A containerized version of the rsyslog utility
...
```

ubi8/ubi コンテナーは、**podman run** で起動した時に他の引数を指定していなければ、bash シェルを実行します。エントリーポイントが設定された場合は、Cmd 値の代わりにその値が使用されます (また、Entriespoint コマンドの引数として Cmd の値が使用されます)。

2 番目の例では、rhel8/rsyslog コンテナーイメージには、組み込みの **install** ラベルおよび **run** ラベルがあります。このラベルは、コンテナーがシステムに設定 (install) され、どのように実行されるか (run) を示します。

2. **コンテナーのマウント** - その内容をさらに調べるには、**podman** コマンドを使用して、アクティブなコンテナーをマウントします。この例では、**rsyslog** コンテナーを実行し、実行中の **rsyslog** コンテナーの一覧を表示して、そのファイルシステムの内容を調べるマウントポイントを表示します。

```
# podman run -d registry.redhat.io/rhel8/rsyslog
# podman ps
CONTAINER ID IMAGE            COMMAND       CREATED       STATUS      PORTS
NAMES
1cc92aea398d ...rsyslog:latest /bin/rsyslog.sh 37 minutes ago Up 1 day ago      myrsyslog
# podman mount 1cc92aea398d
/var/lib/containers/storage/overlay/65881e78.../merged
# ls /var/lib/containers/storage/overlay/65881e78*/merged
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr
var
```

podman mount を行うと、ホストに一覧表示したディレクトリーから、コンテナーのコンテンツにアクセスできます。**ls** を使用して、イメージの内容を調べます。

3. **イメージのパッケージ一覧の確認** - コンテナーにインストールされているパッケージを確認するには、**rpm** コマンドで、コンテナーのマウントポイントにインストールされているパッケージを調べます。

```
# rpm -qa --root=/var/lib/containers/storage/overlay/65881e78.../merged
redhat-release-server-7.6-4.el7.x86_64
filesystem-3.2-25.el7.x86_64
basesystem-10.0-7.el7.noarch
ncurses-base-5.9-14.20130511.el7_4.noarch
glibc-common-2.17-260.el7.x86_64
nspr-4.19.0-1.el7_5.x86_64
libstdc++-4.8.5-36.el7.x86_64
```

2.3.3. リモートイメージの検証

お使いのシステムに取得する前にコンテナイメージを検証する場合は、**skopeo inspect** コマンドを使用できます。**skopeo inspect** で、リモートコンテナーレジストリーにあるイメージに関する情報を表示できます。

次のコマンドは、Red Hat レジストリーから、**ubi8-init** イメージを検証します。

```
# skopeo inspect docker://registry.redhat.io/ubi8/ubi8-init
{
```

```
    "Name": "registry.redhat.io/ubi8/ubi8-init",
    "Digest": "sha256:53dfe24...",
    "RepoTags": [
        "8.0.0-9",
        "8.0.0",
        "latest"
    ],
    "Created": "2019-05-13T20:50:11.437931Z",
    "DockerVersion": "1.13.1",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.access.redhat.com",
        "build-date": "2019-05-13T20:49:44.207967",
        "com.redhat.build-host": "cpt-0013.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "ubi8-init-container",
        "description": "The Red Hat Enterprise Linux Init image is designed to be...
```

2.4. タグ付けイメージ

イメージに名前を追加して、イメージに含まれる内容を理解しやすいものにすることができます。また、タグ付けイメージを使用して、そのイメージが使用するターゲットレジストリーを指定することもできます。**podman tag** コマンドを使用して、複数のパーツから構成されるイメージにエイリアスを追加します。これには、以下が含まれます。

registryhost/username/NAME:tag

必要に応じて **NAME** だけを追加できます。以下に例を示します。

```
# podman tag 474ff279782b myrhel8
```

この例では、**rhel8** イメージのイメージ ID は 474ff279782b です。 **podman tag** を使用すると、イメージ ID には名前 **myrhel8** も付きます。したがって、このコンテナーは、名前 (rhel8 または myrhel8) またはイメージ ID で実行できます。名前に :tag を追加せずに、:latest がタグとして割り当てられていることに注意してください。以下のように、タグを 8.0 に設定している可能性があります。

```
# podman tag 474ff279782b myrhel8:8.0
```

名前の先頭に、オプションでユーザー名やレジストリー名を追加できます。ユーザー名は、実際には、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 上のリポジトリーです。レジストリー名を使用してイメージにタグ付けする方法は、本書の「タグ付けイメージ」セクションで説明します。ユーザー名を追加する例を以下に示します。

```
# podman tag 474ff279782b jsmith/myrhel8
# podman images | grep 474ff279782b
rhel8           latest  474ff279782b  7 days ago  139.6 MB
myrhel8         latest  474ff279782b  7 months ago  139.6 MB
myrhel8         7.1     474ff279782b  7 months ago  139.6 MB
jsmith/myrhel8  latest  474ff279782b  7 months ago  139.6 MB
```

ここでは、1つのイメージ ID に割り当てられているすべてのイメージ名を確認できます。

2.5. イメージの保存および読み込み

ローカルに保存したコンテナーイメージを保存する場合は、**podman save** を使用してイメージをアーカイブファイルまたはディレクトリーに保存し、後で別のコンテナー環境に復元できます。保存するアーカイブは、さまざまなコンテナーイメージ形式、具体的には docker-archive、oci-archive、oci-dir (oci manifext タイプのディレクトリー)、または docker-dir (v2s2 マニフェストタイプのディレクトリー) のいずれかになります。イメージを保存したら、保存したり他の人に送信したり、画像を **読み込んで** 後で再利用したりできます。デフォルトの docker-archive 形式でイメージを tarball として保存する例を次に示します。

```
# podman save -o myrsyslog.tar registry.redhat.io/rhel8/rsyslog:latest
# file myrsyslog.tar
myrsyslog.tar: POSIX tar archive
```

myrsyslog.tar ファイルは、現在のディレクトリーに保存されます。その後、tarball をコンテナーイメージとして再利用する準備ができたら、以下のように別の podman 環境へインポートできます。

```
# podman load -i myrsyslog.tar
# podman images
REPOSITORY                          TAG    IMAGE ID      CREATED     SIZE
registry.redhat.io/rhel8/rsyslog latest 1f5313131bf0  7 weeks ago 235 MB
```

保存 および **読み込み** を使用してイメージを保存して再読み込みする代わりに、**podman export** および **podman import** を使用してコンテナーのコピーを作成できます。

2.6. イメージの削除

システムにあるイメージの一覧を表示するには、**podman 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、**podman rmi** コマンドに、イメージ ID または名前をオプションとして追加して実行します。イメージを削除する前に、イメージから実行したコンテナーを停止する必要があります。 以下に例を示します。

```
# podman rmi ubi8-init
7e85c34f126351ccb9d24e492488ba7e49820be08fe53bee02301226f2773293
```

同じコマンドラインで複数のイメージを削除できます。

```
# podman rmi registry.redhat.io/rhel8/rsyslog support-tools
46da8e23fa1461b658f9276191b4f473f366759a6c840805ed0c9ff694aa7c2f
85cfba5cd49c84786c773a9f66b8d6fca04582d5d7b921a308f04bb8ec071205
```

すべてのイメージを削除する場合は、以下のようなコマンドを使用すると、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前に、コマンドの実行内容を必ず確認してください)。

```
# podman rmi -a
1ca061b47bd70141d11dcb2272dee0f9ea3f76e9afd71cd121a000f3f5423731
ed904b8f2d5c1b5502dea190977e066b4f76776b98f6d5aa1e389256d5212993
83508706ef1b603e511b1b19afcb5faab565053559942db5d00415fb1ee21e96
```

複数の名前 (タグ) が関連付けられているイメージを削除するには、force オプションを追加して削除する必要があります。以下に例を示します。

```
# podman rmi -a
A container associated with containers/storage, i.e. via Buildah, CRI-O, etc., may be associated with
```

```
this image: 1de7d7b3f531

# podman rmi -f 1de7d7b3f531
1de7d7b3f531...
```

第3章 コンテナーの使用

コンテナーは、圧縮されたファイルから展開したコンテナーイメージにあるファイルから生成した、実行中または停止したプロセスを表します。本セクションでは、コンテナーを実行し、そのコンテナーを使用するツールを説明します。

3.1. コンテナーの実行

podman run コマンドを実行すると、コンテナーイメージから起動して新しいコンテナーを作成できます。**podman run** コマンドラインに渡すコマンドは、コンテナーの内部を実行環境と見なします。そのため、デフォルトでは、ホストシステムはほとんど確認できません。たとえば、デフォルトで実行中のアプリケーションからは、以下が確認できます。

- コンテナーイメージが提供するファイルシステム
- コンテナー内からの新規プロセステーブル (ホストからのプロセスは確認できません)

コンテナーに利用できるホストのディレクトリーを作成したり、コンテナーからホストにネットワークポートをマッピングしたり、コンテナーが使用できるメモリーの量を制限したり、コンテナーで利用できる CPU 共有を拡張したりする場合は、**podman run** コマンドラインで実行できます。ここでは、さまざまな機能を有効にする **podman run** コマンドラインの例をいくつか紹介します。

例 1 (クイックコマンドを実行) - この podman コマンドは、 **cat /etc/os-release** コマンドを実行して、コンテナの基盤として使用されているオペレーティングシステムの種類を確認します。コンテナーがコマンドを実行すると、コンテナーは終了し、削除されます (**--rm**)。

```
# podman run --rm registry.redhat.io/ubi8/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.0 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.0"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.0 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.0:latest"
HOME_URL="https://www.redhat.com/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.0
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.0"
...
```

例 2 (コンテナーの Dockerfile を表示) - これは、ホストからコンテナーの内容を検証するためのクイックコマンドを実行する例です。Red Hat Provides には、**/root/buildinfo** が組み込まれている Dockerfile があります。この場合は、ホストからボリュームをマウントする必要がありません。

```
# podman run --rm \
    registry.access.redhat.com/rhel8/rsyslog \
    ls /root/buildinfo
Dockerfile-rhel8-rsyslog-8
```

これで Dockerfile が呼び出されたことが分かったため、その内容を一覧表示できます。

```
# podman run --rm registry.access.redhat.com/rhel8/rsyslog \
    cat /root/buildinfo/Dockerfile-rhel8-rsyslog-8
FROM sha256:eb205f07ce7d0bb63bfe560...
LABEL maintainer="Red Hat, Inc."

RUN INSTALL_PKGS="\
rsyslog \
rsyslog-gnutls \
rsyslog-gssapi \
rsyslog-mysql \
rsyslog-pgsql \
rsyslog-relp \
" && yum -y install $INSTALL_PKGS && rpm -V --nosize
    --nofiledigest --nomtime --nomode $INSTALL_PKGS && yum clean all
LABEL com.redhat.component="rsyslog-container"
LABEL name="rhel8/rsyslog"
LABEL version="8.0"
...
```

例 #3 (コンテナーでシェルを実行) - コンテナーを使用して bash シェルを起動することで、コンテナーを調べ、その内容を変更できます。これにより、コンテナーの名前は、**mybash** に設定されます。**-i** は対話型セッションを作成し、**-t** で端末セッションを開きます。**-i** を使用しないと、シェルが開き、そして終了します。**-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ません。

コマンドを実行すると、シェルプロンプトが表示され、コンテナーから実行コマンドを開始できます。

```
# podman run --name=mybash -it registry.redhat.io/ubi8/ubi /bin/bash
[root@ed904b8f2d5c/]#  yum install procps-ng
[root@ed904b8f2d5c/]#  ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 00:46 pts/0    00:00:00 /bin/bash
root        35     1  0 00:51 pts/0    00:00:00 ps -ef
[root@49830c4f9cc4/]# exit
```

コンテナーを終了すると実行しなくなりますが、新しいソフトウェアパッケージがインストールされた状態で残ります。**podman ps -a** を実行して、コンテナーの一覧を表示します。

```
# podman ps -a
CONTAINER ID IMAGE                COMMAND   CREATED        STATUS                PORTS NAMES
IS INFRA
1ca061b47bd7 .../ubi8/ubi:latest   /bin/bash 8 minutes ago  Exited 12 seconds ago       musing_brown
false
...
```

podman start コマンドに **-ai** オプションを付けると、コンテナーを再度起動できます。以下に例を示します。

```
# podman start -ai mybash
[root@ed904b8f2d5c/]#
```

例 4 (ログファイルのマウントをバインド) - ホストシステムで利用可能なコンテナーからログメッセージを作成する 1 つの方法は、コンテナーでホストの /dev/log デバイスをバインドマウントすることです。この例は、**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する方法を示してい

ます。ここでは、ログメッセージ (この場合は単なる logger コマンド) を生成し、そのメッセージを、ホストから、コンテナーにマウントされている /dev/log デバイスに転送します。**--rm** オプションは、実行後にコンテナーを削除します。

```
# podman run --name="log_test" -v /dev/log:/dev/log --rm \
    registry.redhat.io/ubi8/ubi logger "Testing logging to the host"
# journalctl -b | grep Testing
Nov 12 20:00:10 ubi8 root[17210]: Testing logging to the host
```

例 5 (静的 IP アドレスを持つデーモンとしてサービスを実行） - 以下の例は、**rsyslog** サービスをデーモンプロセスとして実行するため、バックグラウンドで継続的に実行します。また、**podman** に対して、コンテナーのネットワークインターフェースを特定の IP アドレス (10.88.0.44 など) に設定するよう指示します。その後、**podman inspect** コマンドを実行して、IP アドレスが適切に設定されていることを確認します。

```
# podman run -d --ip=10.88.0.44 registry.access.redhat.com/rhel7/rsyslog
efde5f0a8c723f70dd5cb5dc3d5039df3b962fae65575b08662e0d5b5f9fbe85
# podman inspect efde5f0a8c723 | grep 10.88.0.44
            "IPAddress": "10.88.0.44",
```

3.2. 実行中のコンテナーおよび停止したコンテナーの調査

実行中のコンテナーをいくつか作成したら、**podman ps** コマンドを実行して、実行中のコンテナーと、終了または停止したコンテナーの両方を一覧表示できます。**podman inspect** を使用して、そのコンテナー内の特定の情報を確認できます。

3.2.1. コンテナーの一覧表示

たとえば、ホストで実行しているコンテナーがあるとします。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、以下のコマンドを試します。

podman ps - ps オプションは、現在実行中のコンテナーをすべて表示します。

```
# podman run -d registry.redhat.io/rhel8/rsyslog
# podman ps
CONTAINER ID IMAGE              COMMAND         CREATED       STATUS           PORTS NAMES
74b1da000a11 rhel8/rsyslog /bin/rsyslog.sh 2 minutes ago Up About a minute       musing_brown
```

実行していないものの削除されていない (--rm オプション) コンテナーが存在する場合は、コンテナーがハングしているため再起動できます。**podman ps -a** コマンドは、実行中または停止中のコンテナーをすべて表示します。

```
# podman ps -a
CONTAINER ID IMAGE         COMMAND    CREATED    STATUS              PORTS NAMES     IS
INFRA
d65aecc325a4 ubi8/ubi      /bin/bash  3 secs ago Exited (0) 5 secs ago peaceful_hopper false
74b1da000a11 rhel8/rsyslog rsyslog.sh 2 mins ago Up About a minute     musing_brown    false
```

3.2.2. コンテナーの検証

既存のコンテナーのメタデータを検証するには、**podman inspect** コマンドを実行します。コンテナーのメタデータをすべて表示することも、一部のメタデータのみを表示することもできます。たとえば、一部のコンテナーのメタデータをすべて表示するには、以下のコマンドを入力します。

```
# podman inspect 74b1da000a11
    ...
  "ID": "74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2",
  "Created": "2018-11-13T10:30:31.884673073-05:00",
  "Path": "/bin/rsyslog.sh",
  "Args": [
      "/bin/rsyslog.sh"
  ],
  "State": {
      OciVersion": "1.0.1-dev",
      Status": "running",
      Running": true,
    ...
```

inspect を使用して、コンテナーから特定の情報を取り出すこともできます。その情報は階層構造で保存されます。そのため、コンテナーの IP アドレス (NetworkSettings の下の IPAddress) を確認するには、**--format** オプションと、コンテナーの ID を使用します。以下に例を示します。

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' 74b1da000a11
10.88.0.31
```

検証するその他の情報の例としては、.Path (コンテナーで実行するコマンドの表示)、.Args (コマンドへの引数)、.Config.ExposedPorts (コンテナーから公開される TCP ポートまたは UDP ポート)、.State.Pid (コンテナーのプロセス ID の表示)、.HostConfig.PortBindings (コンテナーからホストへのポートマッピング) などがあります。以下は、.State.Pid と .State.StartedAt の例です。

```
# podman inspect --format='{{.State.Pid}}' 74b1da000a11
19593
# ps -ef | grep 19593
root     19593 19583  0 10:30 ?        00:00:00 /usr/sbin/rsyslogd -n
# podman inspect --format='{{.State.StartedAt}}' 74b1da000a11
2018-11-13 10:30:35.358175255 -0500 EST
```

最初の例では、ホストシステム (PID 19593) でコンテナーにした実行ファイルのプロセス ID を確認できます。**ps -ef** コマンドは、それが実行中の rsyslogd デーモンであることを確認します。2 つ目の例は、コンテナーが実行した日時を示しています。

3.2.3. コンテナーのコンテンツを調べる

実行中のコンテナーのコンテンツを調べるには、**podman exec** コマンドを使用できます。**podman exec** を使用し、(**/bin/bash** などの) コマンドを実行して実行中のコンテナープロセスに入り、そのコンテナを調べます。

bash シェルでコンテナーを起動するだけでなく、**podman exec** を使用する理由は、目的のアプリケーションを実行しているときにコンテナーを調査できるためです。コンテナーが目的のタスクを実行しているときに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妨害せずに、コンテナーが実際に何をするかをより的確に確認することができます。

以下は、**podman exec** を使用して、実行中の rsyslog を調べ、そのコンテナーの中身を調べる例です。

1. **コンテナーの起動** - 上述した rsyslog コンテナーイメージなどのコンテナーを起動します。**podman ps** を実行して、コンテナーが実行していることを確認します。

```
# podman ps
CONTAINER ID   IMAGE          COMMAND          CREATED      STATUS       PORTS
```

```
NAMES
74b1da000a11   rsyslog:latest "/usr/rsyslog.sh   6 minutes ago Up 6 minutes          rsyslog
```

2. **podman exec** - でコンテナーを入力します。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、コンテナーの属性を調べます。以下に例を示します。

```
# podman exec -it 74b1da000a11 /bin/bash
[root@74b1da000a11 /]# cat /etc/redhat-release
Red Hat Enterprise Linux release 8.0
[root@74b1da000a11 /]# yum install procps-ng
[root@74b1da000a11 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 15:30 ?        00:00:00 /usr/sbin/rsyslogd -n
root         8     0  6 16:01 pts/0    00:00:00 /bin/bash
root        21     8  0 16:01 pts/0    00:00:00 ps -ef
[root@74b1da000a11 /]# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          39G  2.5G   37G   7% /
tmpfs            64M     0   64M   0% /dev
tmpfs           1.5G  8.7M  1.5G   1% /etc/hosts
shm              63M     0   63M   0% /dev/shm
tmpfs           1.5G     0  1.5G   0% /sys/fs/cgroup
tmpfs           1.5G     0  1.5G   0% /proc/acpi
tmpfs           1.5G     0  1.5G   0% /proc/scsi
tmpfs           1.5G     0  1.5G   0% /sys/firmware
[root@74b1da000a11 /]# uname -r
4.18.0-80.1.2.el8_0.x86_64
[root@74b1da000a11 /]# rpm -qa | more
redhat-release-8.0-0.44.el8.x86_64
filesystem-3.8-2.el8.x86_64
basesystem-11-5.el8.noarch
ncurses-base-6.1-7.20180224.el8.noarch
...
bash-4.2# free -m
              total        used        free      shared  buff/cache   available
Mem:           1941         560         139          10        1241        1189
Swap:          1023          15        1008
[root@74b1da000a11 /]# exit
```

bash シェルからコマンドを実行 (コンテナー内で実行) すると、以下のような複数が項目を分かります。

- コンテナーを、RHEL リリース 8.0 イメージから構築している。
- プロセステーブル (ps -ef) が、/usr/sbin/rsyslogd コマンドのプロセス ID が 1 であることを示している。
- ホストのプロセステーブルで実行しているプロセスは、コンテナーからは確認できない。ただし、rsyslogd プロセスはホストのプロセステーブルで確認できます (ホスト上のプロセス ID は 19593)。
- コンテナーで別のカーネルが実行していない (uname -r はホストシステムのカーネルを示している)。

- rpm -qa コマンドを使用すると、コンテナーに含まれる RPM パッケージを確認できる。つまり、コンテナーに RPM データベースがあります。
- メモリーを表示 (free -m) を表示すると、ホストで利用可能なメモリーが表示される (ただし、コンテナーが実際に使用できるものは、cgroup を使用して制限可能)。

3.3. コンテナーの起動および停止

コンテナーを実行し、削除しなかった場合 (**--rm**)、そのコンテナーはローカルシステムに保存され、再び実行する準備ができています。以前に実行し、削除していない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。

3.3.1. コンテナーの起動

対話的に実行する必要がないコンテナーは、停止後、**start** オプションと、コンテナー ID または名前だけで再起動できます。以下に例を示します。

```
# podman start myrhel_httpd
myrhel_httpd
```

ローカルシェルからコンテナーを起動して作業できるようにするには、-a (attach) オプションおよび -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、exit と入力してシェルを強制終了し、コンテナーを停止します。

```
# podman start -a -i agitated_hopper
[root@d65aecc325a4 /]#  exit
```

3.3.2. コンテナーの停止

端末セッションに割り当て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。以下に例を示します。

```
# podman stop 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

stop オプションは、SIGTERM シグナルを送り、実行中のコンテナーを終了します。コンテナが猶予期間 (デフォルトでは 10 秒) を超えても停止しない場合は、**podman** が SIGKILL シグナルを送ります。また、**podman kill** コマンドを使用して、コンテナーを強制終了 (SIGKILL) するか、コンテナーに別のシグナルを送信します。以下は、SIGHUP シグナルをコンテナーに送信する例です (アプリケーションでサポートされていると、SIGHUP により、アプリケーションが設定ファイルを再読み取りします)。

```
# podman kill --signal="SIGHUP" 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

3.4. コンテナーの削除

システムでハングしているコンテナの一覧を表示するには、**podman ps -a** コマンドを実行してください。不要になったコンテナーを削除するには、**podman rm** コマンドに、コンテナー ID または名前をオプションとして指定して使用します。コンテナーを削除する前に、実行中のコンテナーを停止する必要があります。以下に例を示します。

```
# podman 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。

```
# podman 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、(イメージではなく) すべてのコンテナーをローカルシステムから削除します (この操作を行う前に、コマンドの実行内容を必ず確認してください)。

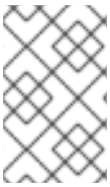
```
# podman rm -a
56c496350bd534da7620fe2fa660526a6fc7f1c57b0298291cd2210311fe723b
83ad58c17b20f9e8271171f3023ae094dbfab6ce5708344a68feb121916961ca
a93b696a1f5629300382a8ce860c4ba42f664db98101e82c2dbcc2074b428faf
bee71e61b53bd8b036b2e8cb8f570ef8308403502760a27ee23a4b675d92b93d
```

第4章 RED HAT UNIVERSAL BASE イメージの使用 (標準、最小、ランタイム)

Red Hat Enterprise Linux (RHEL) ベースイメージは、コンテナーイメージのベースとして使用できます。RHEL 8 では、すべての Red Hat ベースイメージを、新しい Universal Base Images (UBI) として利用できます。これには、自由に利用でき、再配布可能な、RHEL の標準、最小、init、および Red Hat Software Collections のバージョンが含まれます。RHEL ベースイメージは、以下のとおりです。

- **サポートされる** - コンテナー化されたアプリケーションでの使用は、Red Hat によりサポートされています。Red Hat Enterprise Linux で、安全で、テストされ、認定されたものと同じソフトウェアパッケージが含まれています。
- **カタログ化される** - 「Red Hat Container Catalog」 の一覧に追加されます。ここでは、各イメージの説明、技術詳細、および正常指数を確認できます。
- **更新される** - 明確に定義された更新スケジュールで提供されるため、最新のソフトウェアを入手していることが明確になります (「Red Hat Container Image Updates」 を参照)。
- **追跡される** - エラータにより追跡され、各更新の変更を理解するのに役立ちます。
- **再利用可能**: ベースイメージを一度実稼働環境にダウンロードしてキャッシュする必要があります。各ベースイメージは、基盤として含まれるすべてのコンテナーで再利用できます。

RHEL 8 用の UBI は、コンテナーイメージを構築するための、UBI 以外の先行オペレーション (**rhel6**、**rhel7**、**rhel-init**、および **rhel-minimal** のベースイメージ) と同じ品質の RHEL ソフトウェアを提供しますが、それがどのように使用され配布されるかについては、より多くの自由が提供されています。



注記

利用可能な Red Hat UBI イメージの一覧と、UBI リポジトリーおよびソースコードに関連する情報は、「Universal Base Images (UBI): Images, repositories, and packages」 を参照してください。

4.1. RED HAT のベースイメージの概要

Red Hat では、カスタマイズ用のイメージに使用するために、複数のベースイメージを提供しています。RHEL 8 イメージはすべて UBI イメージです。これは、イメージを自由に取得して再配布できることを意味します。このイメージは、Red Hat Registry (registry.access.redhat.com および registry.redhat.io) から入手できます (「Red Hat Container Catalog」 で説明されています)。

RHEL 8 では、標準、最小、および init のベースイメージが利用できます。Red Hat は、「Application Streams」 に基づく一連の言語ランタイムイメージも提供します。これは、特定のランタイムを必要とするアプリケーションのコンテナーを作成する際に構築できます。ランタイムイメージには python、php、ruby、nodejs などが含まれます。

RHEL 7 イメージのセットがあり、RHEL 8 システムで実行できます。RHEL 7 には、UBI (再配布可能) と UBI 以外 (サブスクリプションアクセスが必要で、配布不可) のベースイメージがあります。このようなイメージには、通常のベースイメージ (**rhel7**、**rhel-init**、および **rhel-minimal**) と、UBI イメージ (**ubi7**、 **ubi7-init**、および **ubi7-minimal**) がそれぞれ 3 つ含まれます。

Red Hat は、RHEL 6 システムでコンテナーを実行するツールを提供しませんが、使用可能な RHEL 6 コンテナーイメージを提供します。RHEL 6 で利用できるベースイメージには、スタンダート (**rhel6**) と init (**rhel6-init**) がありますが、最小の RHEL 6 イメージはありません。同様に、RHEL 6 UBI イメージはありません。

4.1.1. 標準の Red Hat ベースイメージの使用

標準の RHEL 8 ベースイメージ (**ubi8**) には、次のような堅牢なソフトウェア機能があります。

- **init system** - systemd サービスの管理に必要な systemd 初期化システムのすべての機能は、標準のベースイメージで利用できます。この init システムを使用すると、Web サーバー (**httpd**) や FTP サーバー (**vsftpd**) などのサービスを自動的に起動するように事前設定された RPM パッケージをインストールできます。

- **yum** - ソフトウェアパッケージをインストールするのに必要とされるソフトウェアは、標準の **yum** コマンドセットから提供されます (**yum**、**yum-config-manager**、**yumdownloader** など)。UBI ベースイメージでは、yum リポジトリーに自由にアクセスして、ソフトウェアを追加および更新できます。

- **ユーティリティー** - 標準ベースイメージには、コンテナーで機能する便利なユーティリティーが含まれています。このベースイメージにあり、最小イメージにはないユーティリティーには、**tar**、**dmidecode**、**gzip**、**getfacl** (およびその他の acl コマンド)、**dmsetup** (およびその他のデバイスマッパーコマンド) が含まれます。

4.1.2. 最小 Red Hat ベースイメージの使用

ubi8-minimal イメージは、余計な部分を取り除いた RHEL イメージで、必要最小限のベースイメージが必要な場合に使用します。大規模な Red Hat エコシステムとして使用できる最小ベースイメージを探している場合は、この最小イメージからで始めることができます。

RHEL の最小イメージは、標準イメージの半分未満のサイズである独自のコンテナーイメージのベースを提供します。ただし、RHEL ソフトウェアリポジトリーに描画し、ソフトウェアが提供するコンプライアンス要件を維持することができます。

以下は、最小ベースイメージの機能の一部です。

- **サイズが小さい** - 最小イメージは、ディスクでは約 92M、圧縮では 32M です。これにより、サイズが、標準イメージの半分に満たなくなります。

- **ソフトウェアインストール (microdnf)** - ソフトウェアリポジトリーおよび RPM ソフトウェアパッケージを使用する完全な **yum** 機能を追加する代わりに、最小イメージには **microdnf** ユーティリティーが含まれます。**microdnf** は、**dnf** の縮小バージョンです。これには、リポジトリーの有効化および無効化と、パッケージのインストール、削除、および更新に必要なもののみが含まれます。また、パッケージのインストール後にキャッシュをクリーンアップする、clean オプションもあります。

- **RHEL パッケージに基づいている** - 最小イメージには通常の RHEL ソフトウェア RPM パッケージが含まれており、追加の言語ファイルやドキュメントなど、いくつかの機能が削除されるため、イメージ構築には引き続き RHEL リポジトリーを使用できます。これにより、RHEL ソフトウェアに基づくコンプライアンス要件を維持できます。最小イメージの機能は、RHEL で実行するアプリケーションの試行に最適であり、オーバーヘッドの量が必要に小さくなります。最小イメージでは実現できないものは、初期化およびサービスの管理システム (systemd または System V init)、Python ランタイム環境、および大量の一般的なシェルユーティリティーです。

- **microdnf のモジュールはサポート対象外** - **dnf** コマンドで使用されるモジュールにより、利用可能な場合は、同じソフトウェアの複数のバージョンをインストールできます。最小イメージに含まれている **microdnf** ユーティリティーは、モジュールに対応していません。したがって、モジュールが必要な場合は、**yum** を含む最小ではないベースイメージを使用する必要があります。

ただし、オペレーティングシステムから要件が少ない単純なバイナリーまたはパッケージ済みのソフトウェアの実行を試みるだけの場合は、最小イメージが必要になることがあります。アプリケーションで RHEL の他のソフトウェアに依存関係がある場合は、**microdnf** を使用してビルド時に必要なパッケージをインストールできます。

Red Hat では、常に最小イメージの最新版が使用されることを前提としています。これは **ubi8**/**ubi-minimal** または **ubi8-minimal**を要求することを意味します。Red Hat では、今後、以前のバージョンの最小イメージをサポートする予定はありません。

4.1.3. Init Red Hat ベースイメージの使用

UBI の **ubi8-init** イメージには、systemd 初期化システムが含まれているため、Web サーバーやファイルサーバーなどの systemd サービスを実行するイメージを構築するのに役立ちます。Init イメージの内容は、標準イメージで得られるものよりも少なくなりますが、最小イメージよりも多くなります。

注記

ubi8-init イメージは **ubi8** イメージの上に構築されるため、その内容はほとんど同じです。ただし、重要な相違点がいくつかあります。**ubi8-init** では、Cmd は **bash**ではなく **/sbin/init** に設定され、デフォルトで systemd Init サービスを開始します。これには **ps** およびプロセス関連のコマンド (**procps-ng** パッケージ) が含まれていますが、ubi8 には含まれていません。また、**ubi8-init** では **SIGRTMIN+3** を **StopSignal** として設定しています。これは、**ubi8-init** の systemd が通常の終了信号 (**SIGTERM** および **SIGKILL**) を無視しているためですが、**SIGRTMIN+3** を受け取った場合は無効になります。

これまでは、Red Hat Enterprise Linux のベースコンテナーイメージは、エンタープライズアプリケーションを実行する目的で設計されていましたが、再配布する予定はありませんでした。したがって、アプリケーションを再配布する必要がある組織によっては、課題が発生する可能性があります。これにより、Red Hat Universal Base イメージが登場しました。

4.2. 各 UBI イメージの相違点

UBI イメージが作成されているため、無料で共有およびデプロイできる公式の Red Hat ソフトウェアの基盤でコンテナーイメージを構築できます。技術的観点では、これはレガシーの Red Hat Enterprise Linux イメージとほぼ同一です。つまり、セキュリティー、パフォーマンス、およびライフサイクルに優れていますが、別のエンドユーザーライセンス契約でリリースされています。以下は、Red Hat UBI イメージの特徴を一部紹介します。

- **RHEL コンテンツのサブセットから構築** - Red Hat Universal Base イメージは、通常の Red Hat Enterprise Linux コンテンツのサブセットから構築されます。選択した UBI イメージを構築するのに使用する内容はすべて、一般に公開された一連の yum リポジトリーでリリースされます。これにより、追加のパッケージをインストールしたり、UBI ベースイメージのパッケージを更新したりできます。

- **再配布可能** - UBI イメージの目的は、Red Hat のお客様、パートナー、ISV などのユーザーが、1 つのコンテナベースイメージを標準とすることで、配信ルールではなくアプリケーションのニーズに重点的に取り込むことができるようにすることです。このイメージは、イメージを実行できるすべての環境で共有および実行できます。基本的なガイドラインに従う限り、UBI ベースのイメージを自由に再配布できます。

- **ベースイメージとランタイムイメージ** - 3 種類のベースイメージのほかに、さまざまなランタイムイメージの UBI バージョンも利用可能です。このランタイムイメージは、python、php、nodejs、ruby などの標準かつサポートされるランタイムから利点を得ることができるアプリケーションの基盤を提供します。

- **有効にした yum リポジトリー** - 以下の yum リポジトリーは、各 RHEL 8 UBI イメージで有効になっています。
  - **ubi-8-baseos** リポジトリーは、コンテナーに追加できる RHEL パッケージの再配布可能なサブセットを保持します。
  - **ubi-8-appstream** リポジトリーは、特定のランタイムを必要とするアプリケーションで使用する環境を標準化するために、UBI イメージに追加できる Red Hat Software Collections パッケージを保持しています。
- **ライセンス** - 「Red Hat Universal Base Image End User License Agreement」 に従い、UBI イメージを自由に使用および再配布できます。
- **UBI RPM の追加** - 事前設定された UBI リポジトリーから UBI イメージに RPM パッケージを追加できます。切断した環境でこのような機能を使用するには、その機能を使用する UBI コンテンツ配信ネットワーク (https://cdn-ubi.redhat.com) をホワイトリストに追加する必要があります。詳細は「Red Hat Container Images are trying to connect to https://cdn-ubi.redhat.com 」を参照してください。

従来の RHEL 7 ベースイメージは引き続きサポートされますが、今後は UBI イメージを使用することが推奨されます。そのため、本章の後続のセクションでは、例に RHEL 8 UBI イメージを使用します。

4.3. UBI イメージの取得

利用可能な Red Hat Universal Base Images の詳細は、「Universal Base Images (UBI): Images, repositories, packages, and source code」を参照してください。

4.4. UBI イメージの取得 (プル)

podman、**buildah**、**skopeo** などのツールで使用できるように、UBI イメージをシステムに取得するには、以下のコマンドを実行します。

```
# podman pull registry.access.redhat.com/ubi8/ubi:latest
# podman pull registry.access.redhat.com/ubi8/ubi-minimal:latest
```

お使いのシステムでイメージが利用可能かどうかを確認するには、以下のコマンドを入力します。

```
# podman images
REPOSITORY                                  TAG    IMAGE ID      CREATED      SIZE
registry.access.redhat.com/ubi8/ubi-minimal  latest  c94a444803e3  8 hours ago  80.9 MB
registry.access.redhat.com/ubi8/ubi          latest  40b488f87628  17 hours ago 214 MB
```

この方法で取得すると、イメージは、**podman**、**buildah**、**skopeo**、および CRI-O コンテナーイメージで利用できるようになりますが、Docker サービスまたは **docker** コマンドには利用できません。Docker でこのイメージを使用する場合は、代わりに **docker pull** コマンドを実行します。

4.5. UBI イメージの再配布

UBI イメージを取得したら、そのイメージを自身のレジストリーにアップロードして、他のレジストリーと共有できます。UBI yum リポジトリーから、必要に応じてイメージをアップグレードまたは追加できます。UBI イメージを、自身のリポジトリーまたは別のサードパーティーのリポジトリーにアップロードする例を以下に示します。

```
# podman pull registry.redhat.io/ubi8/ubi
# podman tag registry.access.redhat.com/ubi8/ubi registry.example.com:5000/ubi8/ubi
# podman push registry.example.com:5000/ubi8/ubi
```

このイメージを使用する方法には制限がいくつかありますが、その方法を参照する方法にもいくつかの制限があります。たとえば、Red Hat Partner Connect Program と、Red Hat Container Certification または Red Hat OpenShift Operator Certification のいずれかで認定されていなければ、Red Hat に認定、または Red Hat にサポートされるとは見なされません。

4.6. UBI イメージの実行

UBI イメージからコンテナーを開始し、(内部から検索できるように) そのイメージで bash シェルを実行するには、以下のコマンドを実行します (完了したら exit と入力します)。

```
# podman run --rm -it registry.access.redhat.com/ubi8/ubi-minimal:latest /bin/bash
[root@da9213157c51 /]#
# podman run --rm -it registry.access.redhat.com/ubi8/ubi:latest /bin/bash
bash-4.2#
```

コンテナーでは、以下のようになります。

- **rpm -qa** を実行して、各コンテナー内のパッケージの一覧を表示します。
- **yum list available** と入力し、UBI yum リポジトリーからイメージに追加できるパッケージを確認します。(**ubi-minimal** コンテナーでは **yum**コマンドが使用できません。)
- 同じ章で後述する「UBI コンテナーイメージのソースコードの取得」で説明しているように、ソースコードを取得します。

Docker サービスを含むシステムでは、代わりに **docker run** を使用できます。

4.7. 実行中の UBI コンテナーにソフトウェアを追加

UBI イメージは、Red Hat のコンテンツだけで構築されます。また、この UBI イメージは、UBI でインストールするのに自由に利用できる Red Hat Enterprise Linux パッケージのサブセットも提供します。ソフトウェアを追加または更新する場合、UBI イメージは、公式の Red Hat の RPM を保持するのに利用できる無料の yum リポジトリーを指定するように事前設定されています。

UBI コンテナーを実行するパッケージを UBI リポジトリーから追加するには、以下を行います。

- **ubi** イメージで、**yum** コマンドをインストールするとパッケージを取り出せます。
- **ubi-minimal** イメージには、**yum** の代わりに、**microdnf** コマンド (小規模な機能セット) が含まれます。

実行中のコンテナーでソフトウェアパッケージを直接インストールして作業する場合は、パッケージを一時的に追加します。もしくは、リポジトリーについて学ぶことになります。UBI ベースのイメージを構築する詳細な方法は「UBI ベースのイメージの構築」を参照してください。

UBI コンテナーにソフトウェアを追加する場合は、サブスクライブしている RHEL ホスト、またはサブスクライブしていない (または非 RHEL) システムで UBI イメージを更新する手順が異なります。UBI イメージを使用する 2 つの方法を以下に示します。

4.7.1. UBI コンテナーへのソフトウェアの追加 (サブスクライブされたホスト)

登録およびサブスクライブされた RHEL ホストで UBI コンテナーを実行している場合は、主要な RHEL Server リポジトリーが、標準の UBI コンテナーとすべての UBI リポジトリーで有効になります。そのため、Red Hat パッケージの全セットが利用できます。UBI の最小コンテナーでは、すべての UBI リポジトリーがデフォルトで有効になりますが、デフォルトではホストではリポジトリーが有効になっていません。

4.7.2. 標準の UBI コンテナーへのソフトウェアの追加

構築するコンテナーが再配布されるようにするには、ソフトウェアを追加するときに、標準の UBI イメージで非 UBI の yum リポジトリーを無効にします。UBI リポジトリー以外の yum リポジトリーをすべて無効にする場合は、ソフトウェアを追加するときに、自由に利用可能なリポジトリーのパッケージのみが使用されます。

サブスクライブしている RHEL ホストから、標準の UBI ベースイメージコンテナー (**ubi8/ubi**) でシェルを開いた状態で、次のコマンドを実行して、そのコンテナーにパッケージ (**bzip2** パッケージなど) を追加します。

```
# yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos bzip2
```

RHEL サーバーのリポジトリーにあり UBI リポジトリーにない標準の UBI コンテナーにソフトウェアを追加するには、リポジトリーを無効にせずに、パッケージをインストールします。

```
# yum install zsh
```

標準の UBI コンテナーから、別のホストのリポジトリーにあるパッケージをインストールするには、必要なリポジトリーを明示的に有効にする必要があります。以下に例を示します。

```
# yum install --enablerepo=rhel-7-server-optional-rpms zsh-html
```

警告

Red Hat UBI リポジトリーに含まれていない Red Hat パッケージをインストールすると、サブスクライブしたホスト外にコンテナーを配信できる範囲が制限される可能性があります。

4.7.3. 最小 UBI コンテナー内へのソフトウェアの追加

UBI の yum リポジトリーは、デフォルトで UBI 最小イメージで有効になります。

UBI 最小コンテナーでサブスクライブしている RHEL ホストのいずれかの UBI yum リポジトリーから、上記と同じパッケージ (**bzip2**) をインストールするには、以下を入力します。

```
# microdnf install bzip2
```

(UBI yum リポジトリーに含まれない) サブスクライブしたホストで利用可能なリポジトリーの最小 UBI コンテナーにパッケージをインストールするには、このリポジトリーを明示的に有効にする必要があります。以下に例を示します。

```
# microdnf install --enablerepo=rhel-7-server-rpms zsh
# microdnf install --enablerepo=rhel-7-server-rpms \
      --enablerepo=rhel-7-server-optional-rpms zsh-html
```

警告

UBI 以外の RHEL リポジトリーを使用して、UBI イメージにパッケージをインストールすると、そのイメージを共有し、サブスクライブしている RHEL システム以外で実行する機能が制限される可能性があります。

4.7.4. UBI コンテナーへのソフトウェアの追加 (サブスクライブしていないホスト)

サブスクライブしていない RHEL ホストまたはその他の Linux システムにある実行中のコンテナーにソフトウェアパッケージを追加するには、yum リポジトリーを無効にする必要がありません。以下に例を示します。

```
# yum install bzip2
```

UBI の最小コンテナーから、サブスクライブしていない RHEL ホストにそのパッケージをインストールする場合は、次のコマンドを実行します。

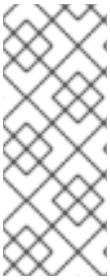
```
# microdnf install bzip2
```

前述のように、実行中の UBI コンテナーにソフトウェアを追加するこの両方の手段は、永続的な UBI ベースのコンテナーイメージを作成することを目的としているわけではありません。そのため、次のセクションで説明するように、UBI イメージに新しいレイヤーを構築する必要があります。

4.7.5. UBI ベースのイメージの構築

UBI ベースのコンテナーイメージは、他のイメージを構築するのと同じ方法で構築できますが、例外があります。再配布できる Red Hat ソフトウェアのみがイメージに含まれるようにするには、イメージを実際にビルドするときに非 UBI yum リポジトリーをすべて無効にする必要があります。

これは、**buildah** ユーティリティーを使用して、Dockerfile から、UBI ベースの Web サーバーコンテナーを作成する例です。



注記

ubi8/ubi-minimal イメージの場合は、以下のように、**yum** の代わりに **microdnf** を使用します。

```
RUN microdnf update -y && rm -rf /var/cache/yum
RUN microdnf install httpd -y && microdnf clean all
```

1. **Dockerfile の作成** - 以下の内容を含む **Dockerfile** を、新しいディレクトリーに追加します。

```
FROM registry.access.redhat.com/ubi8/ubi
USER root
```

```
LABEL maintainer="John Doe"
# Update image
RUN yum update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos
-y && rm -rf /var/cache/yum
RUN yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos
httpd -y && rm -rf /var/cache/yum
# Add default Web page and expose port
RUN echo "The Web Server is Running" > /var/www/html/index.html
EXPOSE 80
# Start the service
CMD ["-D", "FOREGROUND"]
ENTRYPOINT ["/usr/sbin/httpd"]
```

2. **新しいイメージの構築** - そのディレクトリーにいるときに、**buildah** を使用して、新しい UBI レイヤー構造のイメージを作成します。

```
# buildah bud -t johndoe/webserver .
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:latest
STEP 2: USER root
STEP 3: LABEL maintainer="John Doe"
STEP 4: RUN yum update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-
8-baseos -y
. . .
No packages marked for update
STEP 5: RUN yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-
8-baseos httpd -y
Loaded plugins: ovl, product-id, search-disabled-repos
Resolving Dependencies
--> Running transaction check
================================================================
 Package                Arch             Version                      Repository                  Size
================================================================
Installing:
 httpd              x86_64 2.4.37-10
                                    latest-rhubi-8.0-appstream 1.4 M
Installing dependencies:
 apr                x86_64 1.6.3-9.el8        latest-rhubi-8.0-appstream 125 k
 apr-util           x86_64 1.6.1-6.el8        latest-rhubi-8.0-appstream 105 k
 httpd-filesystem   noarch 2.4.37-10
                                    latest-rhubi-8.0-appstream  34 k
 httpd-tools        x86_64 2.4.37-10.
...

Transaction Summary
...
Complete!
STEP 6: RUN echo "The Web Server is Running" > /var/www/html/index.html
STEP 7: EXPOSE 80
STEP 8: CMD ["-D", "FOREGROUND"]
STEP 9: ENTRYPOINT ["/usr/sbin/httpd"]
STEP 10: COMMIT
...
Writing manifest to image destination
Storing signatures
--> 36a604cc0dd3657b46f8762d7ef69873f65e16343b54c63096e636c80f0d68c7
```

3. **テスト** - UBI レイヤー構造の webserver イメージをテストします。

```
# podman run -d -p 80:80 johndoe/webserver
bbe98c71d18720d966e4567949888dc4fb86eec7d304e785d5177168a5965f64
# curl http://localhost/index.html
The Web Server is Running
```

4.7.6. AppStream ランタイムイメージの使用

Red Hat Enterprise Linux 8 AppStream は、コンテナービルドのベースとして使用できる別のコンテナーイメージセットを提供します。このイメージは、RHEL 標準ベースイメージに基づいて構築されており、ほとんどはすでに UBI イメージとして更新されています。このイメージには、特定のランタイム環境で使用する追加ソフトウェアが含まれます。

PHP ランタイムソフトウェアなどが必要な複数のイメージを構築することが予想される場合は、PHP ソフトウェアコレクションイメージから始めることで、そのイメージに対してより一貫性のあるプラットフォームを提供できます。

以下は、UBI ベースイメージに構築された AppStream コンテナーイメージの例です。これは、Red Hat レジストリー (registry.access.redhat.com または registry.redhat.io) から入手できます。

- **ubi8/php-72** - アプリケーションを構築して実行する PHP 7.2 プラットフォーム
- **ubi8/nodejs-10** - アプリケーションを構築して実行する Node.js 10 プラットフォーム。Node.js 10 Source-To-Image ビルドで使用
- **ubi8/ruby25** - アプリケーションを構築して実行する Ruby 2.5 プラットフォーム
- **ubi8/python-27** - アプリケーションを構築して実行する Python 2.7 プラットフォーム
- **ubi8/python-36** - アプリケーションを構築して実行する Python 3.6 プラットフォーム
- **ubi8/s2i-core** - perl、python、ruby などのビルダーイメージのベースとして使用する、基本的なライブラリーおよびツールを含むベースイメージ
- **ubi8/s2i-base** - Source-to-Image ビルドのベースイメージ

このような UBI イメージコンテナーには、従来のイメージと同じ基本ソフトウェアが含まれているため、詳細は『Using Red Hat Software Collections Container Images』 を参照してください。このようなイメージを取得するには、UBI イメージ名を使用してください。

RHEL 8 AppStream コンテナーイメージは、RHEL 8 ベースイメージが更新されるたびに更新されます。RHEL 7 の場合、このイメージ (Red Hat Software Collections イメージと呼ばれています) は、(Dotnet および DevTools の関連イメージである) RHEL ベースイメージの更新とは別のスケジュールで更新されます。このようなイメージの詳細は「Red Hat Container Catalog」 を検索します。更新スケジュールの詳細は「Red Hat Container Image Updates」 を参照してください。

4.7.7. UBI コンテナーイメージソースコードの取得

すべての Red Hat UBI ベースのイメージのソースコードは、ダウンロード可能なコンテナーの形で入手できます。続行する前に、Red Hat ソースコンテナーに注意してください。

- コンテナーとしてパッケージ化されているにもかかわらず、ソースコンテナーイメージを実行できません。システムに Red Hat ソースコンテナーイメージをインストールするには、**podman pull** コマンドを使用する代わりに、**skopeo** コマンドを使用します。

- **skopeo copy** コマンドを使用して、ソースコンテナーイメージをローカルシステムのディレクトリーにコピーします。
- **skopeo inspect** コマンドを使用して、ソースコンテナーイメージを検査します。

- **skopeo** コマンドの詳細は、セクション 1.5.「Using skopeo to work with container registries」を参照してください。
- ソースコンテナーイメージは、それらが表すバイナリーコンテナーに基づいて名前が付けられます。たとえば、ある標準的な RHEL UBI 8 コンテナーでは、**registry.access.redhat.com/ubi8:8.1-397** に **-source** を追加してソースコンテナーイメージを取得します (**registry.access.redhat.com/ubi8:8.1-397-source**)。
- ソースコンテナーイメージがローカルディレクトリーにコピーされたら、**tar** コマンド、**gzip** コマンド、および **rpm** コマンドの組み合わせを使用して、そのコンテンツを操作できます。
- コンテナーイメージがリリースされてから、関連するソースコンテナーが利用可能になるまでに数時間かかる場合があります。

手順

1. **skopeo copy** コマンドを使用して、ソースコンテナーイメージをローカルディレクトリーにコピーします。

```
$ skopeo copy \
docker://registry.access.redhat.com/ubi8:8.1-397-source \
dir:$HOME/TEST
...
Copying blob 477bc8106765 done
Copying blob c438818481d3 done
Copying blob 26fe858c966c done
Copying blob ba4b5f020b99 done
Copying blob f7d970ccd456 done
Copying blob ade06f94b556 done
Copying blob cc56c782b513 done
Copying blob dcf9396fdada done
Copying blob feb6d2ae2524 done
Copying config dd4cd669a4 done
Writing manifest to image destination
Storing signatures
```

2. **skopeo inspect** コマンドを使用して、ソースコンテナーイメージを検査します。

```
$ skopeo inspect dir:$HOME/TEST
{
    "Digest":
"sha256:7ab721ef3305271bbb629a6db065c59bbeb87bc53e7cbf88e2953a1217ba7322",
    "RepoTags": [],
    "Created": "2020-02-11T12:14:18.612461174Z",
    "DockerVersion": "",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:1ae73d938ab9f11718d0f6a4148eb07d38ac1c0a70b1d03e751de8bf3c2c87fa",
        "sha256:9fe966885cb8712c47efe5ecc2eaa0797a0d5ffb8b119c4bd4b400cc9e255421",
```

```
        "sha256:61b2527a4b836a4efbb82dfd449c0556c0f769570a6c02e112f88f8bbcd90166",
        ...
        "sha256:cc56c782b513e2bdd2cc2af77b69e13df4ab624ddb856c4d086206b46b9b9e5f",
        "sha256:dcf9396fdada4e6c1ce667b306b7f08a83c9e6b39d0955c481b8ea5b2a465b32",

"sha256:feb6d2ae252402ea6a6fca8a158a7d32c7e4572db0e6e5a5eab15d4e0777951e"
    ],
    "Env": null
}
```

3. すべてのコンテンツを展開するには、次のコマンドを実行します。

```
$ cd $HOME/TEST
$ for f in $(ls); do tar xvf $f; done
```

4. 結果を確認するには、次のコマンドを実行します。

```
$ find blobs/ rpm_dir/
blobs/
blobs/sha256
blobs/sha256/10914f1fff060ce31388f5ab963871870535aaaa551629f5ad182384d60fdf82
rpm_dir/
rpm_dir/gzip-1.9-4.el8.src.rpm
```

5. コンテンツの調査と使用を開始します。

4.7.8. UBI イメージの使用に関するヒント

UBI イメージを使用する際に考慮すべき問題を以下に示します。

- 既存の Red Hat Software Collections ランタイムイメージで使用される数百の RPM パッケージは、新しい UBI イメージに同梱されている yum リポジトリーに保存されます。UBI イメージにこの RPM をインストールして、必要なランタイム (python、php、nodejs など) をエミュレートします。

- UBI の最小イメージ (**ubi8/ubi-minimal**) には、いくつかの言語ファイルとドキュメントが含まれていないため、そのコンテナーで **rpm -Va** を実行すると、多くのパッケージの内容が欠落しているか変更されているものとして表示されます。そのコンテナーの中にファイルの完全なリストがあることが重要な場合は、**Tripwire** などのツールを使用してファイルをコンテナーに記録し、後で確認することを考慮してください。

- レイヤー構造のイメージを作成したら、**podman history** を使用して、どの UBI イメージが構築されたかを確認します。たとえば、前に示した webserver の例を完了したら、**podman history johndoe/webserver** と入力して、それが構築されたイメージに、Dockerfile の FROM 行に追加した UBI イメージのイメージ ID が含まれていることを確認します。

4.7.9. UBI の新機能を要求する方法

Red Hat パートナーおよびお客様は、標準の方法でサポートチケットを作成することにより、パッケージリクエストを含む新機能を要求できます。Red Hat 以外のお客様はサポートを受けませんが、適切な RHEL 製品向けの標準 Red Hat Bugzilla でリクエストを作成できます。

「Red Hat Bugzilla Queue」 も併せて参照してください。

4.7.10. UBI のサポートケースを作成する方法

Red Hat パートナーおよびお客様は、サポートされる Red Hat プラットフォーム (OpenShift/RHEL) で UBI を実行している場合に、通常の方法でサポートチケットを作成できます。Red Hat サポートスタッフがパートナーとお客様の問題に対応します。

「サポートケースを作成する」も併せて参照してください。

第5章 特殊なコンテナーイメージの実行

コンテナーおよびコンテナーイメージの一般的な操作方法に慣れたら、本セクションを参照して、役に立つと思われる特定タイプのコンテナーイメージを学習します。これには、以下が含まれます。

- **toolbox** - 問題のデバッグや機能の監視に必要なツールをインストールしてホストシステムに負担をかける代わりに、**toolbox** コマンドを実行できます。Toolbox は、ホストでレポートを実行したり問題を診断したりするために使用できるツールを保持する **support-tools** コンテナーイメージを起動します。

- **Runlabels** - 一部のコンテナーイメージには、あらかじめ設定されたオプションと引数を使用してコンテナーを実行できるラベルが組み込まれています。ランレベルで **podman run** を実行すると、コンテナーイメージのインストール、実行、削除、またはアップグレードを行ったときに、規定の機能セットが得られる可能性があります。

5.1. TOOLBOX でコンテナーホストのトラブルシューティング

toolbox ユーティリティーを使用すればトラブルシューティングツールを Red Hat Enterprise Linux 8 システムに直接インストールする代わりに、このツールを一時的に追加し、完了したら簡単に破棄できます。**toolbox** ユーティリティーは次のように機能します。

- **registry.redhat.io/rhel8/support-tools** イメージをローカルシステムにプルします。

- イメージからコンテナーを起動し、ホストシステムにアクセスできるコンテナー内でシェルを実行します。

support-tools コンテナーを使用すると、以下を行うことができます。

- ホストシステムにインストールされていない可能性のあるコマンド (例: **sosreport**、**strace**、または **tcpdump**) を、ホストシステムで動作できるように実行します。

- ホストシステムで使用するコンテナーに、追加ソフトウェアをインストールします。

- 作業が完了したら、コンテナーを破棄します。

以下は、典型的な **toolbox** セッションを示しています。

手順

1. toolbox パッケージおよび podman パッケージがインストールされていない場合はインストールします。そのために推奨される方法は、コンテナーツールのフルセットをインストールすることです。

```
# yum module install container-tools -y
```

2. toolbox コマンドを実行して、**support-tools** イメージをプルして実行します (プロンプトが表示されたら、Red Hat カスタマーポータルの認証情報を入力します)。

```
# toolbox
Trying to pull registry.redhat.io/rhel8/support-tools...
...
Would you like to authenticate to registry: 'registry.redhat.io' and try again? [y/N] y
Username: johndoe
Password: **************
Login Succeeded!
```

```
Trying to pull registry.redhat.io/rhel8/support-tools...Getting image source signatures
...
Storing signatures
30e261462851238d38f4ef2afdaf55f1f8187775c5ca373b43e0f55722faaf97
Spawning a container 'toolbox-root' with image 'registry.redhat.io/rhel8/support-tools'
Detected RUN label in the container image. Using that as the default...
command: podman run -it --name toolbox-root --privileged --ipc=host --net=host --pid=host -e
HOST=/host -e NAME=toolbox-root -e IMAGE=registry.redhat.io/rhel8/support-tools:latest -v
/run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v /etc/localtime:/etc/localtime -
v /:/host registry.redhat.io/rhel8/support-tools:latest

bash-4.4#
```

bash シェルが開き、コンテナー内でコマンドを実行できる状態になります。

3. コンテナー内から、ホストの root ファイルシステムが **/host** ディレクトリーから利用できます。ここに表示されるその他のディレクトリーは、コンテナーの中にすべて表示されます。

```
# ls /
bin   dev  home  lib lost+found  mnt  proc  run   srv  tmp  var
boot  etc  host  lib64 media     opt  root  sbin  sys  usr
```

4. コンテナー内から、コマンドを試すことができます。たとえば、**sosreport** を実行してシステムに関する情報を生成し、Red Hat サポートに送信できます。

```
bash-4.4# sosreport

sosreport (version 3.6)
This command will collect diagnostic and configuration information from
this Red Hat Enterprise Linux system and installed applications.

An archive containing the collected information will be generated in
/host/var/tmp/sos.u82evisb and may be provided to a Red Hat support
representative.
...
Press ENTER to continue, or CTRL-C to quit.   <Press ENTER>
...
Your sosreport has been generated and saved in:
  /host/var/tmp/sosreport-rhel81beta-12345678-2019-10-29-pmgjncg.tar.xz
The checksum is: c4e1fd3ee45f78a17afb4e45a05842ed
Please send this file to your support representative.
```

sosreport は、コンテナー内であることを認識している点に注意してください。したがって、ホストで実行してレポートをホスト (**/host/var/tmp/sosreport-…**) に保存します。

5. コンテナーにソフトウェアパッケージをインストールして、コンテナーにないツールを追加します。たとえば、ホストで実行中のプロセスのコアダンプを取得するには、**procps** パッケージおよび **gcore** パッケージをインストールし、**ps** を使用して実行中のデーモンのプロセス ID を取得し、**gcore** を使用してコアダンプを取得します。

```
bash-4.4# yum install procps gdb -y
bash-4.4# ps -ef | grep chronyd
994        809     1  0 Oct28 ?        00:00:00 /usr/sbin/chronyd
bash-4.4# gcore -o /host/tmp/chronyd.core 809
Missing separate debuginfo for target:/usr/sbin/chronyd
```

```
Try: dnf --enablerepo='*debug*' install /usr/lib/debug/.build-
id/96/0789a8a3bf28932b093e94b816be379f16a56a.debug
...
Saved corefile /host/tmp/chronyd.core.809
[Inferior 1 (process 809) detached]
# exit
```

exit を入力すると、コンテナーを離れて、ホストに戻ります。ホストの **/host/tmp/chronyd.core.809** に保存されたファイルは、ホストの **/tmp/chronyd.core.809** から利用できることが確認できます。

この時点で、コンテナーは実行していませんが、システム上には存在したままになります。以下を選択できます。

- Start up the container again- **toolbox** を再入力して、(**toolbox-root** という名前の) コンテナーを再起動します。以前コンテナーに行ったソフトウェアの追加や変更は維持されます。
- Start with a fresh container- 古いコンテナーを取り除くには、**podman rm toolbox-root** と入力します。次に、**toolbox** を再実行して、新規の **support-tools** コンテナーで開始します。
- Start with different values- 以下の値をホストの **/root/.toolboxrc** に追加して、**toolbox** で使用するレジストリー、イメージ、またはコンテナー名を変更できます。
  - REGISTRY - toolbox イメージがプルされるレジストリーを変更します。たとえば、**REGISTRY=registry.example.com** です。
  - IMAGE - 使用されるイメージを変更します。たとえば、**IMAGE=mysupport-tools** などです。
  - TOOLBOX_NAME - 実行中のコンテナーに割り当てられた名前を変更します。たとえば、**TOOLBOX_NAME=mytoolbox** となります。

次回の **toolbox** を実行すると、**.toolboxrc** ファイルから新しい値が使用されます。

5.1.1. ホストへの権限の付与

support-tools コンテナー (または特権コンテナー) 内からその他のコマンドを実行すると、非特権コンテナー内で実行した場合と異なる動作をする場合があります。**sosreport** は、コンテナーで実行しているタイミングを指示できますが、その他のコマンドは、ホストシステム (**/host** ディレクトリー) で機能するように特に指示する必要があります。以下は、コンテナーからホストに開くことができる、または開くことができない機能の例です。

- **特権** - 特権コンテナー (**--privileged**) は、デフォルトでホストのアプリケーションを root ユーザーとして実行します。SELinux セキュリティーコンテキスト **unconfined_t** で実行しているため、コンテナーにはこの機能があります。そのため、たとえば、root ユーザーが所有するホストからマウントされたファイルおよびディレクトリーを削除できます。
- **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** セットを使用して) **ps -e** コマンドを実行すると、ホストで実行しているすべてのプロセスを表示できます。そのため、ホストから特権コンテナーで実行する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、コマンドによっては、コンテナーからプロセスにアクセスしようとすると、パーミッションの問題が発生することがあります。
- **ネットワークインターフェース** - デフォルトでは、コンテナーには外部のネットワークインターフェースとループバックネットワークインターフェースが 1 つずつだけあります。ネットワークインターフェースをホストに開く (**--net=host**) と、コンテナーからネットワークイン

ターフェースに直接アクセスできます。

- **プロセス間のコミュニケーション** - 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。したがって、**ipcs** などのコマンドを実行して、ホスト上の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に関する情報を表示できます。

5.2. ランレベルでコンテナーの実行

Red Hat イメージには、そのイメージと連携するために事前に設定されたコマンドラインを提供するラベルが含まれているものがあります。**podman container runlabel <label>** コマンドを使用すると、その <label> で定義されたコマンドをイメージに対して実行するように **podman** に指示できます。既存のラベルには次のものが含まれます。

- **install** - イメージを実行する前にホストシステムを設定します。通常、これにより、後で実行するときにコンテナーがアクセスできるホストにファイルとディレクトリーが作成されます。
- **run** - コンテナーの実行時に使用する podman コマンドラインオプションを特定します。通常、オプションはホストで権限を開き、コンテナーがホストで永続的に維持する必要があるホストのコンテンツをマウントします。
- **uninstall** - コンテナーの実行後にホストシステムを削除します。

1 つ以上のランレベルを持つ Red Hat イメージには、rsyslog および support-tools イメージが含まれます。以下の手順では、このイメージの使用方法を説明します。

5.2.1. ランレベルでの rsyslog の実行

rhel8/rsyslog コンテナーイメージは、rsyslogd デーモンのコンテナー化されたバージョンを実行するように設計されています。rsyslog イメージ内には、ランレベル **install**、**run**、**uninstall** があります。以下の手順に従って、rsyslog イメージのインストール、実行、アンインストールを行います。

手順

1. rsyslog イメージをプルします。

```
# podman pull registry.redhat.io/rhel8/rsyslog
```

2. rsyslog の **install** ランレベルを表示します (ただし実行はしていません)。

```
# podman container runlabel install --display rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/install.sh
```

これは、コマンドがホストに特権を開き、コンテナー内の **/host** にホストの root ファイルシステムをマウントし、**install.sh** スクリプトを実行します。

3. rsyslog の **install** ランレベルを実行します。

```
# podman container runlabel install rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/install.sh
Creating directory at /host//etc/pki/rsyslog
Creating directory at /host//etc/rsyslog.d
```

```
Installing file at /host//etc/rsyslog.conf
Installing file at /host//etc/sysconfig/rsyslog
Installing file at /host//etc/logrotate.d/syslog
```

これにより、rsyslog イメージが後で使用するホストシステムにファイルが作成されます。

4. rsyslog の **run** ランレベルを表示します。

```
# podman container runlabel run --display rhel8/rsyslog
command: podman run -d --privileged --name rsyslog --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e
NAME=rsyslog --restart=always registry.redhat.io/rhel8/rsyslog:latest /bin/rsyslog.sh
```

これは、コマンドがホストへの特権を開き、rsyslog コンテナーを起動して rsyslogd デーモンを実行するときに、コンテナー内のホストから多数のファイルとディレクトリーをマウントすることを示しています。

5. rsyslog の **run** ランレベルを実行します。

```
# podman container runlabel run rhel8/rsyslog
command: podman run -d --privileged --name rsyslog --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e
NAME=rsyslog --restart=always registry.redhat.io/rhel8/rsyslog:latest /bin/rsyslog.sh
28a0d719ff179adcea81eb63cc90fcd09f1755d5edb121399068a4ea59bd0f53
```

rsyslog コンテナーは権限を開き、ホストから必要なものをマウントし、バックグラウンドで rsyslogd デーモンを実行します (-d)。rsyslogd デーモンは、ログメッセージを収集し、メッセージを **/var/log** ディレクトリー内のファイルに送信します。

6. rsyslog の **uninstall** ランレベルを表示します。

```
# podman container runlabel uninstall --display rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/uninstall.sh
```

7. rsyslog の **uninstall** ランレベルを実行します。

```
# podman container runlabel uninstall rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/uninstall.sh
```

この場合、**uninstall.sh** スクリプトは、**/etc/logrotate.d/syslog** ファイルを削除します。設定ファイルはクリーンアップされないことに注意してください。

5.2.2. ランレベルで support-tools の実行

rhel8/support-tools コンテナーイメージは、sosreport や sos-collector などのツールを実行して、ホストシステムを解析するのに役に立ちます。support-tools イメージの実行を簡素化するために、**run** ランレベルが追加されています。以下の手順では、support-tools イメージの実行方法を説明します。

手順

1. support-tools イメージをプルします。

```
# podman pull registry.redhat.io/rhel8/support-tools
```

2. support-tools の **run** ランレベルを表示します (ただし実行はしていません)。

```
# podman container runlabel run --display rhel8/support-tools
command: podman run -it --name support-tools --privileged --ipc=host --net=host --pid=host -
e HOST=/host -e NAME=support-tools -e IMAGE=registry.redhat.io/rhel8/support-tools:latest
-v /run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -v /:/host registry.redhat.io/rhel8/support-tools:latest
```

これは、コマンドがディレクトリーをマウントし、ホストシステムに対して特権と名前空間 (ipc、net、およびpid) を開くことを示しています。ホストの root ファイルシステムを、コンテナーの **/host** ディレクトリーに割り当てます。

3. support-tools に、**run** ランレベルを実行します。

```
# podman container runlabel run rhel8/support-tools
command: podman run -it --name support-tools --privileged --ipc=host --net=host --pid=host -
e HOST=/host -e NAME=support-tools -e IMAGE=registry.redhat.io/rhel8/support-tools:latest
-v /run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -v /:/host registry.redhat.io/rhel8/support-tools:latest
bash-4.4#
```

これにより、support-tools コンテナー内に bash シェルが開きます。

ホストシステム (**/host**) に対してレポートツールまたはデバッグツールを実行できるようになりました。完了したら、**exit** と入力してシェルを終了し、コンテナーを停止します。

第6章 BUILDAH でコンテナーイメージの構築

buildah コマンドを使用すると、作業中コンテナー、Dockerfile、または scratch コンテナーから、コンテナーイメージを作成できます。作成されたイメージは OCI に準拠しているため、「OCI Runtime Specification」 (Docker、CRI-O など) を満たす任意のコンテナーランタイムで動作します。

このセクションでは、**buildah** コマンドを使用して、コンテナーおよびコンテナーイメージを作成し、作業する方法を説明します。

6.1. BUILDAH について

Buildah を使用することは、以下の点で、**docker** コマンドでイメージを作成するのと異なります。

- **デーモンは使用されない** - Docker デーモンが無効になります。したがって、Buildah を使用する場合に、コンテナーランタイム (Docker、CRI-O など) が必要ありません。
- **ベースイメージまたは scratch**- 別のコンテナーに基づいてイメージを構築するだけでなく、空のイメージ (scratch) で開始することもできます。
- **外部のビルドツール** -イメージ自体にはビルドツールが含まれません。その結果、Buildah は以下のようになります。
  - 構築するイメージのサイズは小さくなります。
  - 作成されるイメージに、コンテナーを構築するのに使用されるソフトウェア (gcc、make、yum など) がなく、イメージの安全性が高まります。
  - イメージの転送に必要なリソースが少ないイメージを作成します (サイズが小さくなるため)。

Buildah は、データを個別に保存し、イメージを構築するだけでなく、そのイメージをコンテナーとして実行することで、Docker やその他のコンテナーランタイムを使用せずに動作させることができます。デフォルトでは、Buildah は、イメージを、**containers-storage** (/var/lib/containers) のように識別される領域に保存します。

注記

buildah コマンドがデフォルトで使用するコンテナーの保管場所は、CRI-O コンテナーエンジンが、イメージのローカルコピーを保存するのに使用する場所と同じになります。そのため、CRI-O または Buildah で、もしくは **buildah** コマンドでレジストリーから取得したイメージは、同じディレクトリー構造に保存されます。ただし、現在、CRI-O および Buildah は、イメージを共有することはできますが、コンテナーを共有することはできません。

buildah コマンドと一緒に使用するオプションが多数あります。**buildah** コマンドで可能な主なアクティビティーには、以下のことが含まれます。

- **Dockerfile からコンテナーを構築** - Dockerfile を使用して新規コンテナーイメージ (**buildah bud**) を構築します。
- **別のイメージまたは scratch イメージにコンテナーを構築** - 既存のベースイメージ (**<imagename> から buildah**) または新規 (**scratch の buildah**) から、scratch コンテナーを構築します。

- **コンテナーまたはイメージの検証** - コンテナーまたはイメージ (**buildah inspect**) に関連付けられているメタデータを表示します。
- **コンテナーのマウント** - コンテナーの root ファイルシステムをマウントして、このコンテンツを追加または変更します (**buildah mount**)。
- **新しいコンテナーレイヤーの作成** - コンテナーの root ファイルシステムで更新された内容をファイルシステムレイヤーとして使用して、コンテンツを新しいイメージにコミット (**buildah commit**) します。
- **コンテナーのアンマウント** - マウントされたコンテナーのマウントを解除します (**buildah umount**)。
- **コンテナーまたはイメージの削除** - コンテナー (**buildah rm**) またはコンテナーイメージ (**buildah rmi**) を削除します。

Buildah の詳細は、GitHub Buildah のページ を参照してください。GitHub Buildah サイトには、man ページと、RHEL バージョンで利用できるよりも新しい可能性があるソフトウェアが含まれています。以下は、Buildah に関するその他の記事です。

- Buildah Tutorial 1: Building OCI container images
- Buildah Tutorial 2: Using Buildah with container registries
- Buildah Blocks - Getting Fit

6.1.1. Buildah のインストール

buildah パッケージは、RHEL 8 の container-tools モジュールで利用できます (**yum module install container-tools**)。buildah パッケージを個別にインストールするには、以下のコマンドを実行します。

```
# yum -y install buildah
```

buildah パッケージをインストールすると、buildah パッケージに含まれている man ページで、その使用方法の詳細を確認できます。利用可能な man ページとその他のドキュメントを表示して、man ページを開くには、以下のコマンドを入力します。

```
# rpm -qd buildah
# man buildah
buildah(1)         General Commands Manual         buildah(1)

NAME
 Buildah - A command line tool that facilitates building OCI container images.
...
```

次のセクションでは、**buildah** を使用してコンテナーを取得し、Dockerfile または scratch からコンテナーを構築し、さまざまな方法でコンテナを管理する方法を説明します。

6.2. BUILDAH でイメージの取得

buildah で使用するコンテナーイメージを取得するには、**buildah from** コマンドを使用します。RHEL 8.0 を使用している場合は、リポジトリーへの認証で問題が発生する可能性があることに注意してください。バグ を参照してください。これは、**buildah** コマンドを使用して、Red Hat Registry から、RHEL 8 イメージを作業コンテナーとして取得する方法です。

```
# buildah from registry.redhat.io/ubi8/ubi
Getting image source signatures
Copying blob…
Writing manifest to image destination
Storing signatures
ubi-working-container
# buildah images
IMAGE ID      IMAGE NAME                        CREATED AT         SIZE
3da40a1670b5  registry.redhat.io/ubi8/ubi:latest  May 8, 2019 21:55  214 MB
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME         CONTAINER NAME
c6c9279ecc0f     *     3da40a1670b5 ...ubi8/ubi:latest ubi-working-container
```

buildah from コマンドの結果は、イメージ (registry.redhat.io/ubi8/ubi:latest) と、そのイメージから実行する準備ができている作業コンテナー (ubi-working-container) です。そのコンテナーからコマンドを実行する方法の例を以下に示します。

```
# buildah run ubi-working-container cat /etc/redhat-release
Red Hat Enterprise Linux release 8.0
```

これで、イメージとコンテナーを Buildah で使用できるようになりました。

6.3. BUILDAH で DOCKERFILE からのイメージの構築

buildah コマンドを使用して、Dockerfile から新しいイメージを作成できます。次の手順では、イメージの実行時に実行される簡単なスクリプトを含むイメージを構築する方法を説明します。

この簡単な例では、現在のディレクトリーにある Dockerfile (コンテナーイメージを構築するための命令を保持) と myecho (画面に数単語を表示するスクリプト) の 2 つのファイルで始まります。

```
# ls
Dockerfile  myecho
# cat Dockerfile
FROM registry.redhat.io/ubi8/ubi
ADD myecho /usr/local/bin
ENTRYPOINT "/usr/local/bin/myecho"
# cat myecho
echo "This container works!"
# chmod 755 myecho
# ./myecho
This container works!
```

現行ディレクトリーの Dockerfile で、以下のように新しいコンテナーを構築します。

```
# buildah bud -t myecho .
STEP 1: FROM registry.redhat.io/ubi8/ubi
STEP 2: ADD myecho /usr/local/bin
STEP 3: ENTRYPOINT "/usr/local/bin/myecho"
```

buildah bud コマンドは、myecho という名前の新規イメージを作成します。新しいイメージを表示するには、以下を入力します。

```
# buildah images
IMAGE NAME        IMAGE TAG  IMAGE ID      CREATED AT          SIZE
localhost/myecho  latest     a3882af49784  Jun 21, 2019 12:21  216 MB
```

次に、イメージを実行して、それが機能していることを確認します。

6.3.1. 構築するイメージの実行

以前ビルドしたイメージが機能することを確認するには、**podman run** を使用してイメージを実行します。

```
# podman run localhost/myecho
This container works!
```

6.3.2. Buildah でのコンテナーの検証

buildah inspect を使用して、コンテナやイメージに関する情報を表示できます。たとえば、**myecho** イメージをビルドして検査するには、次のように入力します。

```
# buildah from localhost/myecho
# buildah inspect localhost/myecho | less
{
 "Type": "buildah 0.0.1",
 "FromImage": "docker.io/library/myecho:latest",
 "FromImage-ID": "e2b190ac8...",
 "Config": "{\"created\":\"2018-11-13...

 "Entrypoint": [
     "/usr/local/bin/myecho"
  ],
  "WorkingDir": "/",
  "Labels": {
     "architecture": "x86_64",
     "authoritative-source-url": "registry.access.redhat.com",
     "build-date": "2018-09-19T20:46:28.459833",
```

同じイメージのコンテナーを検証するには、以下を入力します。

```
# buildah inspect myecho-working-container | less
{
    "Type": "buildah 0.0.1",
    "FromImage": "docker.io/library/myecho:latest",
    "FromImage-ID": "e2b190a...",
    "Config": "{\"created\":\"2018-11-13T19:5...
...
    "Container": "myecho-working-container",
    "ContainerID": "c0cd2e494d...",
    "MountPoint": "",
    "ProcessLabel": "system_u:system_r:svirt_lxc_net_t:s0:c89,c921",
    "MountLabel": "",
```

コンテナーの出力には、コンテナー名、コンテナー ID、プロセスラベル、マウントラベルなどの情報がイメージ内のものに追加されました。

6.4. BUILDAH で新規イメージを作成するコンテナーの変更

buildah コマンドを使用して、既存のコンテナーを変更し、この変更を新しいコンテナーイメージに適用する方法はいくつかあります。

- コンテナーをマウントして、そのコンテナーにファイルをコピーします。
- **buildah copy** および **buildah config** を使用して、コンテナーを変更します。

コンテナーを修正したら、**buildah commit** を使用して、新しいイメージに変更を適用します。

6.4.1. buildah mount でコンテナーの変更

buildah from でイメージを取得したあと、そのイメージを新しいイメージの基盤として使用できます。以下のテキストは、作業コンテナーをマウントし、そのコンテナーにファイルを追加して、新しいイメージへの変更をコミットすることで、新しいイメージを作成する方法を示しています。

以前使用した作業コンテナーを表示するには、以下を入力します。

```
# buildah containers
CONTAINER ID BUILDER IMAGE ID     IMAGE NAME  CONTAINER NAME

dc8f21af4a47   *     1456eedf8101 registry.redhat.io/ubi8/ubi:latest
            ubi-working-container
6d1ffccb557d   *     ab230ac5aba3 docker.io/library/myecho:latest
            myecho-working-container
```

コンテナーイメージをマウントし、マウントポイントを変数 ($mymount) に設定すると、処理が容易になります。

```
# mymount=$(buildah mount myecho-working-container)
# echo $mymount
/var/lib/containers/storage/devicemapper/mnt/176c273fe28c23e5319805a2c48559305a57a706cc7ae7bec7da4cd79edd3c02/rootfs
```

マウントしたコンテナーで、以前作成したスクリプトに内容を追加します。

```
# echo 'echo "We even modified it."' >> $mymount/usr/local/bin/myecho
```

(myecho という名前の) 新しいイメージを作成するために追加した内容をコミットするには、以下を入力します。

```
# buildah commit myecho-working-container containers-storage:myecho2
```

新しいイメージに変更が含まれることを確認するには、作業コンテナーを作成して実行します。

```
# buildah images
IMAGE ID     IMAGE NAME     CREATED AT          SIZE
a7e06d3cd0e2 docker.io/library/myecho2:latest
                        Oct 12, 2017 15:15  3.144 KB
# buildah from docker.io/library/myecho2:latest
myecho2-working-container
```

```
# podman run docker.io/library/myecho2
This container works!
We even modified it.
```

そのスクリプトに追加された新しい **echo** コマンドが、追加のテキストを表示するのを確認できます。

終了すると、コンテナーをアンマウントできます。

```
# buildah umount myecho-working-container
```

6.4.2. **buildah copy** および **buildah config** でコンテナーの変更

buildah copy を使用して、最初にマウントしなくても、ファイルをコンテナにコピーできます。以下は、前のセクションで作成 (およびマウントを解除) した **myecho-working-container**を使用して、新しいスクリプトをコンテナーにコピーし、コンテナーの設定を変更して、デフォルトでそのスクリプトを実行します。

newecho という名前のスクリプトを作成し、実行可能にします。

```
# cat newecho
echo "I changed this container"
# chmod 755 newecho
```

新しい作業コンテナーを作成します。

```
# buildah from myecho:latest
myecho-working-container-2
```

newecho を、コンテナーの /usr/local/bin にコピーします。

```
# buildah copy myecho-working-container-2 newecho /usr/local/bin
```

newecho スクリプトを、新しいエントリーポイントとして使用するように設定を変更します。

```
# buildah config --entrypoint "/bin/sh -c /usr/local/bin/newecho "myecho-working-container-2
```

新しいコンテナーを実行すると、**newecho** コマンドが実行します。

```
# buildah run myecho-working-container-2
I changed this container
```

期待したようにコンテナーが動作すれば、新しいイメージ (mynewecho) にコミットできます。

```
# buildah commit myecho-working-container-2 containers-storage:mynewecho
```

6.5. BUILDAH で SCRATCH からイメージを新規作成

ベースイメージから開始する代わりに、中身がない新規コンテナーと、少数のコンテナーメタデータのみを保持する新規コンテナーを作成できます。これは、**scratch** コンテナーと呼ばれます。ここで、**buildah** コマンドを使用して、scratch コンテナーから始まるイメージを作成することを選択するときに、考慮すべきいくつかの問題があります。

- scratch コンテナーでは、scratch イメージに依存関係のない実行ファイルをコピーして、最小コンテナーを機能させるため、いくつかの設定を行うことができます。
- **yum** パッケージや **rpm** パッケージなどのツールを使用して、scratch コンテナーを作成するには、少なくともコンテナー内の RPM データベースを初期化し、リリースパッケージを追加する必要があります。以下の例は、実行方法を示しています。
- たくさんの RPM パッケージを追加した場合は、scratch イメージではなく、ベースイメージの **ubi** または **ubi-minimal** を使用することを検討してください。このベースイメージには、ドキュメント、言語パック、およびその他のコンポーネントが含まれ、結果としてイメージのサイズが小さくなる可能性があります。

この例では、コンテナーに Web サービス (httpd) を追加し、これを実行するように設定します。まず、scratch コンテナーを作成します。

```
# buildah from scratch
working-container
```

これにより、以下のようにマウントできる空のコンテナー (イメージなし) のみが作成されます。

```
# scratchmnt=$(buildah mount working-container)
# echo $scratchmnt
/var/lib/containers/storage/devicemapper/mnt/cc92011e9a2b077d03a97c0809f1f3e7fef0f29bdc6ab5e86b85430ec77b2bf6/rootfs
```

scratch イメージで RPM データベースを初期化し、redhat-release パッケージを追加します (RPM が機能するために必要なその他のファイルが含まれます)。

```
# yum install -y --releasever=8 --installroot=$scratchmnt redhat-release
```

httpd サービスを scratch ディレクトリーにインストールします。

```
# yum install -y --setopt=reposdir=/etc/yum.repos.d \
    --installroot=$scratchmnt \
    --setopt=cachedir=/var/cache/dnf httpd
```

コンテナーの index.html ファイルにテキストを追加すると、後でテストできるようになります。

```
# echo "Your httpd container from scratch worked." > $scratchmnt/var/www/html/index.html
```

init サービスとして httpd を実行する代わりに、**buildah config** オプションをいくつか設定して、httpd デーモンをコンテナーから直接実行するオプションを設定します。

```
# buildah config --cmd "/usr/sbin/httpd -DFOREGROUND" working-container
# buildah config --port 80/tcp working-container
# buildah commit working-container localhost/myhttpd:latest
```

現在のところ、**podman** コマンドを使用して、イメージ ID を使用して新しいイメージをコンテナーとして実行できます。

```
# podman images
REPOSITORY          TAG             IMAGE ID        CREATED          SIZE
localhost/myhttpd   latest          47c0795d7b0e    9 minutes ago    665.6 MB
```

```
# podman run -p 8080:80 -d --name httpd-server 47c0795d7b0e
# curl localhost:8080
Your httpd container from scratch worked.
```

6.6. BUILDAH でイメージまたはコンテナーの削除

特定のコンテナーやイメージの使用を終えたら、それぞれ **buildah rm** または **buildah rmi** で削除します。以下に例をいくつか示します。

前のセクションで作成したコンテナーを削除する場合は、以下を入力して、マウントされたコンテナーを表示し、マウントを解除して削除します。

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
05387e29ab93     *     c37e14066ac7 docker.io/library/myecho:latest  myecho-working-container
# buildah mount
05387e29ab93 /var/lib/containers/storage/devicemapper/mnt/9274181773a.../rootfs
# buildah umount 05387e29ab93
# buildah rm 05387e29ab93
05387e29ab93151cf52e9c85c573f3e8ab64af1592b1ff9315db8a10a77d7c22
```

以前作成したイメージを削除する場合は、以下を入力します。

```
# buildah rmi docker.io/library/myecho:latest
untagged: docker.io/library/myecho:latest
ab230ac5aba3b5a0a7c3d2c5e0793280c1a1b4d2457a75a01b70a4b7a9ed415a
```

6.7. BUILDAH でコンテナーレジストリーの使用

Buildah を使用すると、ローカルシステムと、パブリックまたはプライベートのコンテナーレジストリーとの間で、コンテナーイメージをプッシュまたはプルできます。以下の例では、方法を説明します。

- buildah を使用して、コンテナーをプッシュして、プライベートレジストリーからプルします。
- ローカルシステムと Docker レジストリー間でコンテナーをプッシュし、プルします。
- 認証情報を使用して、コンテナーをプッシュする際にレジストリーアカウントにコンテナーを関連付けます。

buildah コマンドと一緒に skopeo コマンドを使用して、コンテナーイメージの情報に関するレジストリーを問い合わせます。

6.7.1. プライベートレジストリーへのコンテナーのプッシュ

コンテナーをプライベートコンテナーレジストリーにプッシュする **buildah** コマンドは、コンテナーをプッシュする **docker** コマンドと同じように機能します。以下が必要です。

- プライベートレジストリーを設定します (OpenShift のコンテナーレジストリーを使用するか、Red Hat Quay コンテナーレジストリーを設定できます)。
- プッシュするコンテナーイメージを作成または取得します。
- **buildah push** を使用して、イメージをレジストリーにプッシュします。

ローカルの Buildah コンテナーストレージからイメージをプッシュするには、イメージ名を確認してから、**buildah push** コマンドを使用してプッシュします。ローカルイメージ名と、場所を含む新しい名前の両方を忘れないでください。たとえば、TCP ポート 5000 でリッスンしているローカルシステムで実行しているレジストリーは、localhost:5000 として識別されます。

```
# buildah images
IMAGE ID     IMAGE NAME                       CREATED AT          SIZE
cb702d492ee9 docker.io/library/myecho2:latest Nov 12, 2018 16:50     3.143 KB

# buildah push --tls-verify=false myecho2:latest localhost:5000/myecho2:latest
Getting image source signatures
Copying blob sha256:e4efd0...
...
Writing manifest to image destination
Storing signatures
```

curl コマンドを使用して、レジストリー内のイメージを一覧表示し、**skopeo** で、イメージに関するメタデータを検証します。

```
# curl http://localhost:5000/v2/_catalog
{"repositories":["myatomic","myecho2"]}
# curl http://localhost:5000/v2/myecho2/tags/list
{"name":"myecho2","tags":["latest"]}
# skopeo inspect --tls-verify=false docker://localhost:5000/myecho2:latest | less
{
    "Name": "localhost:5000/myecho2",
    "Digest": "sha256:8999ff6050...",
    "RepoTags": [
        "latest"
    ],
    "Created": "2017-11-21T16:50:25.830343Z",
    "DockerVersion": "",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.redhat.io",
```

この時点で、コンテナーレジストリーからコンテナーイメージをプルできるツールは、プッシュしたイメージのコピーを取得できます。たとえば、RHEL 7 システムでは、docker デーモンを起動してイメージをプルすると、以下のように **docker** コマンドを実行してそのイメージを使用できるようにします。

```
# systemctl start docker
# docker pull localhost:5000/myecho2
# docker run localhost:5000/myecho2
This container works!
```

6.7.2. Docker Hub へのコンテナーのプッシュ

buildah コマンドで、Docker Hub の認証情報を使用して、Docker Hub からイメージをプッシュおよびプルできます。この例では、ユーザー名とパスワード (testaccountXX:My00P@sswd) を、自身の Docker Hub 認証情報に置き換えます。

```
# buildah push --creds testaccountXX:My00P@sswd \
    docker.io/library/myecho2:latest docker://testaccountXX/myecho2:latest
```

プライベートレジストリーと同様に、**podman** コマンド、**buildah** コマンド、または **docker** コマンドを使用して、Docker Hub からコンテナーを取得して実行できます。

```
# podman run docker.io/textaccountXX/myecho2:latest
This container works!
# buildah from docker.io/textaccountXX/myecho2:latest
myecho2-working-container-2
# podman run myecho2-working-container-2
This container works!
```

第7章 PODMAN で SYSTEMD サービスとしてコンテナーを実行

Podman (Pod Manager) は、簡単なデーモンレスツールである、完全に機能するコンテナーエンジンです。Podman は、他のコンテナーエンジンからの移行を容易にし、Pod、コンテナー、およびイメージの管理を可能にする Docker-CLI と同等のコマンドラインを提供します。当初は、Linux システム全体の起動や、起動順序、依存関係の確認、失敗したサービスの復元などのサービス管理を行うよう設計されていました。これは、systemd のような、本格的な初期化システムのジョブです。

Red Hat は、コンテナーと systemd の統合について先駆者になり、他のサービスと機能が Linux システムで管理されているのと同じように、Podman により構築された OCI 形式および Docker 形式のコンテナーを管理できます。本章では、systemd 初期化サービスを使用してコンテナーを操作する 2 つの方法を説明します。

- **systemd でコンテナーを起動** - ホストコンピューターに systemd ユニットファイルを設定して、自動的にホストの起動、停止、ステータスの確認を行い、その他の方法ではコンテナーを systemd サービスとして管理できます。
- **systemd を使用してコンテナーでサービスの起動** - 多くの Linux サービス (Web サーバー、ファイルサーバー、データベースサーバーなど) は、Red Hat Enterprise Linux が systemd サービスとして動作するようにパッケージ化されています。最新の RHEL コンテナーイメージを使用している場合は、RHEL コンテナーイメージを設定して systemd サービスを開始し、コンテナーが起動すると、コンテナーで選択したサービスを自動的に開始できます。

次の 2 つのセクションでは、このような方法で systemd コンテナーを使用する方法を説明します。

7.1. SYSTEMD でコンテナーの起動

コンテナーを systemd サービスとして開始するように設定すると、コンテナー化されたサービスの実行順序の定義、依存関係の確認 (別のサービスが実行していること、ファイルが利用可能であること、リソースがマウントされていることなど)、さらに **runc** コマンドを使用してコンテナーが開始するように設定できます。

このセクションでは、RHEL システムで systemd サービスとして直接実行するように設定されたコンテナーの例を示します。systemd サービスファイルの自動生成は、「systemd ユニットファイルの生成」を参照してください。

1. システムで実行するイメージを取得します。たとえば、docker.io の Alpine Linux に基づく最小限のイメージを使用するには、次のコマンドを実行します。

```
# podman pull docker.io/library/alpine:latest
```

2. **~/.config/systemd/user** ディレクトリーに汎用ユニットの構成ファイルを作成して、コンテナーを systemd サービスとして構成します。たとえば、**~/.config/systemd/user/container.service** の内容は次のようになります。

```
# cat ~/.config/systemd/user/container.service
[Unit]
Description=Podman in Systemd

[Service]
Restart=on-failure
ExecStartPre=/usr/bin/rm -f /%t/%n-pid /%t/%n-cid
ExecStart=/usr/bin/podman run --conmon-pidfile  /%t/%n-pid  --cidfile /%t/%n-cid -d
alpine:latest top
ExecStop=/usr/bin/sh -c "/usr/bin/podman rm -f `cat /%t/%n-cid`"
```

```
KillMode=none
Type=forking
PIDFile=/%t/%n-pid

[Install]
WantedBy=multi-user.target
```

- **Restart=on-failure** 行は、再起動ポリシーを設定し、サービスを正常に開始または停止できない場合、またはプロセスがゼロ以外のステータスで終了した場合に、systemd にサービスを再開するように指示します。
- **ExecStart** 行は、コンテナーの開始方法を示しています。
- **ExecStop** 行は、コンテナーを停止して削除する方法を示しています。**top** を実行するバックグラウンドで **alpine:latest** コンテナーを実行できます。**podman run** コマンドには、2 つのコマンドラインオプションがあります。
- **--conmon-pidfile** オプションは、ホストで実行している **conmon** プロセスのプロセス ID を格納するパスを指します。**conmon** プロセスはコンテナーと同じ終了ステータスで終了します。これにより、**systemd** は正しいサービスステータスを報告し、必要に応じてコンテナーを再起動できます。
- **--cidfile** オプションは、コンテナー ID を格納するパスを指します。
- **%t** は、ランタイムディレクトリーのルートへのパスです (例: **/run/user/$UserID**)。
- **%n** は、サービスの完全な名前です。

たとえば、サービス名が **container** で、ユーザー ID が 1000 の場合、上記の構成では、**/run/user/1000/container.service-pid** の **conmon-pidfile** に配置し、**/run/user/1000/container.service-cid** の **cidfile** に配置します。

3. systemd マネージャーの設定を再読み込みするには、次のコマンドを実行します。

```
# systemctl --user daemon-reload
```

4. サービスを有効にしてシステム起動時に開始するには、次のコマンドを実行します。

```
# systemctl --user enable container.service
```

5. サービスをすぐに開始してサービスの状態を確認するには、次のコマンドを実行します。

```
# systemctl --user start container.service
# systemctl --user status container.service
* container.service - Podman in Systemd
   Loaded: loaded (/home/valentin/.config/systemd/user/container.service; disabled; vendor
preset: enabled)
   Active: active (running) since Mon 2019-11-18 15:32:56 CET; 1min 5s ago
  Process: 189705 ExecStartPre=/usr/bin/rm -f //run/user/1000/container.service-pid
//run/user/1000/container.service-cid (code=exited, status=0/SUCCESS)
  Process: 189706 ExecStart=/usr/bin/podman run --conmon-pidfile
//run/user/1000/container.service-pid --cidfile //run/user/1000/container.service-cid -d
alpine:latest top (code=exited, status=0/SUCCESS)
 Main PID: 189731 (conmon)
   CGroup: /user.slice/user-1000.slice/user@1000.service/container.service
           ├─189724 /usr/bin/fuse-overlayfs [...]
```

```
        ├─189726 /usr/bin/slirp4netns [...]
        ├─189731 /usr/bin/conmon [...]
        └─189737 top
```

6. 実行中または終了したコンテナーの一覧を表示するには、次のコマンドを実行します。

```
# podman ps
CONTAINER ID  IMAGE                           COMMAND  CREATED      STATUS          PORTS
NAMES
f20988d59920  docker.io/library/alpine:latest  top   12 seconds ago  Up 11 seconds ago
funny_zhukovsky
```

7. **container.service** を停止するには、次のコマンドを実行します。

```
# systemctl --user stop container.service
```

systemd でサービスを設定する方法の詳細は、システム管理者ガイドの章「systemd によるサービス管理」か、「Running containers with Podman and shareable systemd services」 を参照してください。

7.2. SYSTEMD でコンテナーでサービスの起動

systemd 初期化システムを含むパッケージは、公式の Red Hat Enterprise Linux Init ベースイメージである registry.access.redhat.com/ubi8/ubi-init に含まれています。つまり、これは、systemd で管理するために作成されたアプリケーションが、コンテナー内で起動および管理できることを示しています。systemd を実行しているコンテナーは、以下を行います。

- /sbin/init プロセス (systemd サービス) を起動して、コンテナーで PID 1 として実行します。
- コンテナーにインストールされ、有効になっているすべての systemd サービスを、依存関係の順番に起動します。
- systemd により、サービスを再起動したり、コンテナーで開始したサービスのゾンビプロセスを強制終了したりできます。

systemd サービスとして使用する準備ができているコンテナーを構築する一般的な手順は以下のとおりです。

- コンテナーに systemd-enabled サービスを含むパッケージをインストールします。これには、Apache Web Server (httpd) 、FTP サーバー (vsftpd) 、プロキシーサーバー (squid) など、RHEL に同梱されるサービスが多数含まれます。この例では、Apache (httpd) Web サーバーをインストールします。
- httpd パッケージおよび vsftpd パッケージが UBI リポジトリーに含まれています。squid パッケージをインストールするには、RHEL サブスクリプションが必要です。
- systemctl コマンドを使用して、コンテナーでサービスを有効にします。
- コンテナーで使用するサービスのデータを追加します (この例では、Web サーバーのテストページを追加します)。実際のデプロイメントでは、おそらく外部ストレージに接続します。
- サービスへのアクセスに必要なポートを公開します。

この例では、コンテナーをホストシステムで実行している場合に、systemd サービス (/sbin/init) により自動的に起動する Web サーバー (httpd) をインストールして設定する Dockerfile を作成して、コンテナーを構築します。

1. **Dockerfile の作成** - このディレクトリーで、以下の内容を含む Dockerfile という名前のファイルを作成します。

```
FROM registry.access.redhat.com/ubi8/ubi-init
RUN yum -y install httpd; yum clean all; systemctl enable httpd;
RUN echo "Successful Web Server Test" > /var/www/html/index.html
RUN mkdir /etc/systemd/system/httpd.service.d/; echo -e '[Service]\nRestart=always' >
/etc/systemd/system/httpd.service.d/httpd.conf
EXPOSE 80
```

   Dockerfile は、httpd パッケージをインストールし、システムの起動時 (つまりコンテナーが起動した時) に httpd サービスが開始するようにし、テストファイル (index.html) を作成し、Web サーバーをホスト (ポート 80) に公開し、コンテナーが起動すると systemd init サービス (/sbin/init) を開始します。

2. **コンテナーの構築** - Dockerfile を含むディレクトリーから、次のコマンドを入力します。

```
# podman build --format=docker -t mysysd .
```

3. **Selinux のパーミッションを開く** - システムで SELinux が有効になっている場合は、以下に示すように、systemd でコンテナーを実行する **container_manage_cgroup** ブール値を有効にする必要があります (詳細は「Container running systemd fails to run after upgrade to Red Hat Enterprise Linux 7.5 」を参照)。

```
# setsebool -P container_manage_cgroup 1
```

4. **コンテナーの実行** - コンテナーを構築し、mysysd という名前を付けたら、次のコマンドでコンテナーを実行します。

```
# podman run -d --name=mysysd_run -p 80:80 mysysd
```

   このコマンドでは、mysysd イメージが mysysd_run コンテナーをデーモンプロセスとして実行し、コンテナーのポート 80 がホストシステムのポート 80 に公開されます。

5. **コンテナーが実行していることを確認** - コンテナーが実行中で、サービスが機能していることを確認するには、以下のコマンドを実行します。

```
# podman ps | grep mysysd_run
a282b0c2ad3d  localhost/mysysd:latest  /sbin/init  15 seconds ago  Up 14 seconds ago
0.0.0.0:80->80/tcp  mysysd_run
# curl localhost/index.html
Successful Web Server Test
```

この時点で、Web サーバーをコンテナーの systemd サービスとして起動するコンテナーがあります。Dockerfile を変更し、必要に応じてデータを設定し、ポートを開いて、同様の方法で任意のサービスをインストールおよび実行します。

第8章 コンテナーのコマンドライン参照

8.1. PODMAN

podman コマンド (Pod Manager の略) を使用すると、Kubernetes、Docker ランタイム、またはその他のコンテナーランタイムを使用せずに、コンテナーをスタンドアロンエンティティーとして実行できます。それは、**docker** コマンドの代わりに機能できるツールです。同じコマンドライン構文を実装しながら、さらにコンテナーの管理機能を追加します。**podman** 機能は次のとおりです。

- docker **インターフェースに基づいている** - **podman** 構文は **docker** コマンドに似ているため、**docker** に精通している場合には、**podman** への移行が簡単です。
- **コンテナーおよびイメージの管理** - 以下を行うために、Docker 互換と OCI 互換の両方のコンテナーイメージを、**podman** とともに使用できます。
  - コンテナーの実行、停止、および再起動
  - コンテナーイメージの作成および管理(プッシュ、コミット、設定、ビルドなど)
- **Pod の管理** - 個々のコンテナーを実行する以外に、**podman** は、Pod にグループ化された一連のコンテナーを実行できます。Pod は、Kubernetes が管理するコンテナーの最小単位です。
- **ランタイムなしでの動作** - ランタイム環境は、コンテナーと連携するため、**podman** では使用されません。

以下は、知っておくべき **podman** の実装機能です。

- Podman、Buildah、および CRI-O のコンテナーエンジンはすべて、Docker の保存場所 (**/var/lib/docker**) をデフォルトで使用する代わりに、同じバックエンドストアディレクトリー **/var/lib/containers** を使用します。
- Podman、Buildah、および CRI-O は、同じストレージディレクトリーを共有しているため、互いのコンテナーと対話することはできません。ただし、このツールはイメージを共有できます。また、この機能は、コンテナーを共有できます。
- **docker** などの **podman** コマンドは、Dockerfile からコンテナーイメージを構築できます。
- **podman** コマンドは、**CRI-O** サービスが利用できない場合に有益なトラブルシューティングツールです。
- **podman** が対応していない **docker**コマンドのオプションには、network、node、plugin (**podman** はプラグインをサポートしません)、rename (rm および create を使用して **podman** でコンテナーの名前を変更します)、secret、service、stack、swarm (**podman** は Docker Swarm をサポートしません) が含まれます。container および imageオプションは、**podman** で直接使用されるサブコマンドを実行するのに使用します。
- プログラムで **podman** と対話するには、Podman のリモート API が、varlink と呼ばれる技術を使用して利用できます。これにより、**podman** は、(RHEL 8 Web コンソールや **atomic** コマンドなど) リモートツールからの API 要求をリッスンし、それに応答します。

8.1.1. podman コマンドの使用

docker コマンドを使用してコンテナーを操作するのに慣れている場合は、そのほとんどの機能とオプションが **podman** の機能とオプションに一致することが分かるでしょう。表 1 は、**podman** で使用できるコマンドの一覧です (**podman -h** を実行するとこの一覧を表示できます)。

表8.1 podman が対応するコマンド

podman コマンド	説明	podman コマンド	説明
attach	実行中のコンテナーに割り当てる	**commit**	変更したコンテナーから新規イメージを作成する
build	Dockerfile 命令を使用してイメージを構築する	**create**	コンテナーを作成するが、起動はしない
diff	コンテナーのファイルシステムの変更を検証する	**exec**	実行中のコンテナーでプロセスを実行する
export	コンテナーのファイルシステムの内容を tar アーカイブとしてエクスポートする	**help, h**	コマンド一覧、または特定のコマンドのヘルプを表示する
history	指定したイメージの履歴を表示する	**images**	ローカルストレージ内のイメージを一覧表示する
import	tarball をインポートして、ファイルシステムイメージを作成する	**info**	システム情報を表示する
inspect	コンテナーまたはイメージの設定を表示する	**kill**	1 つ以上の実行中のコンテナーに特定のシグナルを送信する
load	アーカイブからイメージを読み込む	**login**	コンテナーレジストリーにログインする
logout	コンテナレジストリーからログアウトする	**logs**	コンテナーのログを取得する
mount	作業中のコンテナーの root ファイルシステムをマウントする	**pause**	1 つ以上のコンテナー内の全プロセスを一時停止する
ps	コンテナーの一覧を表示する	**port**	コンテナーのポートマッピングまたは特定のマッピングを一覧表示する
pull	レジストリーからイメージを取得する	**push**	指定した宛先にイメージをプッシュする
restart	1 つ以上のコンテナーを再起動する	**rm**	ホストから 1 つ以上のコンテナーを削除する。実行している場合は、**-f** を追加する

rmi	ローカルストレージから 1 つ以上のイメージを削除する	**run**	新しいコンテナーでコマンドを実行する
save	イメージをアーカイブに保存する	**search**	イメージのレジストリーを検索する
start	1 つ以上のコンテナーを起動する	**stats**	1 つ以上のコンテナーの CPU、メモリー、ネットワーク I/O、ブロック I/O、および PID の割合を表示する
stop	1 つ以上のコンテナーを停止する	**tag**	ローカルイメージに名前を追加する
top	コンテナーの実行中のプロセスを表示する	**umount, unmount**	作業コンテナーのルートファイルシステムをアンマウントする
unpause	1 つ以上のコンテナーでプロセスの一時停止を解除する	**version**	podman のバージョン情報を表示する
wait	1 つ以上のコンテナーをブロックする		

8.1.2. 基本的な podman コマンドの試行

podman は、**docker** コマンドの機能と構文を反映しています。このオプションを使用してコンテナーと連携する例は、「Working with Docker Formatted Container Images」 を参照してください。ほとんどの場合は、**docker** を **podman** に置き換えます。以下は、**podman** の使用例です。

8.1.3. ローカルシステムにコンテナーイメージを取得 (プル)

```
# podman pull registry.redhat.io/ubi8/ubi
Trying to pull registry.redhat...Getting image source signatures
Copying blob sha256:d1fe25896eb5cbcee...
Writing manifest to image destination
Storing signatures
fd1ba0b398a82d56900bb798c...
```

8.1.4. ローカルコンテナーイメージの一覧表示

```
# podman images
REPOSITORY                TAG     IMAGE ID      CREATED      SIZE
registry.redhat.io/ubi8/ubi  latest  de9c26f23799  5 weeks ago  80.1MB
registry.redhat.io/ubi8/ubi  latest  fd1ba0b398a8  5 weeks ago  211MB
```

8.1.5. コンテナーイメージの検証

```
# podman inspect registry.redhat.io/ubi8/ubi | less
[
    {
        "Id": "4bbd153adf8487a8a5114af0d6...",
        "Digest": "sha256:9999e735605c73f...",
        "RepoTags": [
            "registry.access.redhat.com/ubi8/ubi:latest"
        ],
        "RepoDigests": [
            "registry.access.redhat.com/ubi8/ubi/rhel@sha256:9999e7356...
```

8.1.6. コンテナーイメージの実行

コンテナーでシェルを開くコンテナーを実行します。

```
# podman run -it registry.redhat.io/ubi8/ubi /bin/bash
[root@8414218c04f9 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 13:48 pts/0    00:00:00 /bin/bash
root        21     1  0 13:49 pts/0    00:00:00 ps -ef
[root@8414218c04f9 /]# exit
#
```

8.1.7. 実行中または終了したコンテナーを一覧表示

```
# podman ps -a
CONTAINER ID   IMAGE                                        COMMAND
   CREATED AT                     STATUS                  PORTS NAMES
440becd26893   registry.redhat.io/ubi8/ubi-minimal:latest  /bin/bash
   2018-05-10 09:02:52 -0400 EDT   Exited (0) About an hour ago  happy_hodgkin
8414218c04f9   registry.redhat.io/ubi8/ubi:latest          /bin/bash
   2018-05-10 09:48:07 -0400 EDT   Exited (0) 14 minutes ago     nostalgic_boyd
```

8.1.8. コンテナーまたはイメージの削除

コンテナー ID を使用して、コンテナーを削除します。

```
# podman rm 440becd26893
```

イメージ ID または名前を使用して、コンテナーイメージを削除します (-f を使用して強制します)。

```
# podman rmi registry.redhat.io/ubi8/ubi
# podman rmi de9c26f23799
# podman rmi -f registry.redhat.io/ubi8/ubi:latest
```

8.1.9. Kube pod yaml ファイルの生成

podman generate コマンドを使用して、コンテナーまはは Pod から Kubernetes Pod yaml ファイルを作成します。

1. デーモンプロセス (この例では mariadb) として実行するコンテナー化されたサービスを開始します。

```
# podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass \
    -e MYSQL_DATABASE=db -p 3306:3306 --name mymariadb rhscl/mariadb-102-rhel7
```

2. コンテナー名または ID を取得します。

```
# podman ps
CONTAINER ID  IMAGE
   COMMAND              CREATED         STATUS
       PORTS                 NAMES
e421a3424ab0  registry.access.redhat.com/rhscl/mariadb-102-rhel7:latest
   container-entrypo...  19 seconds ago  Up 16 seconds ago
       0.0.0.0:3306->3306/tcp  mymariadb
```

3. コンテナー名または ID を使用して Kubernetes yaml 出力を生成し、これをファイルに転送します。

```
# podman generate kube mymariadb > mymariadbkube.yaml
# less mymariadbkube.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-10-31T13:19:41Z"
  labels:
    app: mymariadb
  name: mymariadb
spec:
  containers:
  - command:
...
 - name: MYSQL_VERSION
      value: "10.2"
    - name: CONTAINER_SCRIPTS_PATH
      value: /usr/share/container-scripts/mysql
    - name: MYSQL_PREFIX
      value: /opt/rh/rh-mariadb102/root/usr
    - name: ENABLED_COLLECTIONS
      value: rh-mariadb102
    - name: DESCRIPTION
      value: MariaDB is a multi-user, multi-threaded SQL database server. The container
        image provides a containerized packaging of the MariaDB mysqld daemon and
        client application. The mysqld server daemon accepts connections from clients
        and provides access to content from MariaDB databases on behalf of the clients.
 - name: STI_SCRIPTS_PATH
      value: /usr/libexec/s2i
    - name: PROMPT_COMMAND
      value: . /usr/share/container-scripts/mysql/scl_enable
    - name: MYSQL_USER
      value: user
    - name: MYSQL_PASSWORD
      value: pass
...
```

podman generate コマンドは、コンテナーに接続されている論理ボリュームマネージャー (LVM) の論理ボリュームまたは物理ボリュームを反映しません。

4. yaml ファイルを使用して、Kubernetes または OpenShift 環境で Pod を作成できます。

```
# kubectl create -f mymariadbkube.yaml
```

8.1.10. systemd ユニットファイルの生成

Podman により、**systemd** はコンテナープロセスを制御および管理できます。**podman generate systemd** コマンドを使用して、非 root (ルートレス) ユーザーとして既存のコンテナーの systemd ユニットファイルを生成できます。

1. コンテナーを作成します (例: **myfedora**)。

```
# podman create -d --name myfedora fedora:latest top
54502f309f3092d32b4c496ef3d099b270b2af7b5464e7cb4887bc16a4d38597
```

2. コンテナー名または ID を使用して systemd ユニットファイルを生成し、それを **~/.config/systemd/user/container-myfedora.service** ファイルに送ります。

```
# podman generate systemd --name myfedora > ~/.config/systemd/user/container-
myfedora.service
container-myfedora.service
autogenerated by Podman 1.6.2
Tue Nov 19 15:49:15 CET 2019

[Unit]
Description=Podman container-myfedora.service
Documentation=man:podman-generate-systemd(1)

[Service]
Restart=on-failure
ExecStart=/usr/bin/podman start myfedora
ExecStop=/usr/bin/podman stop -t 10 myfedora
KillMode=none
Type=forking
PIDFile=/run/user/1000/overlay-
containers/54502f309f3092d32b4c496ef3d099b270b2af7b5464e7cb4887bc16a4d38597/userd
ata/conmon.pid

[Install]
WantedBy=multi-user.target
```

3. これで、コンテナーの状態を開始して確認できます。

```
# systemctl --user start container-myfedora.service
# systemctl --user status container-myfedora.service
```

4. コンテナーを停止できます。

```
# systemctl --user stop container-myfedora.service
```

詳細は、「Running containers with Podman and shareable systemd services」 を参照してください。

8.1.11. コンテナーの SELinux ポリシーの作成

コンテナーの SELinux ポリシーを生成するには、UDICA ツールを使用します。詳細は「udica の SELinux ポリシージェネレーターの概要」を参照してください。

8.1.12. MPI での podman の使用

Podman を Open MPI (Message Passing Interface) と共に使用して、HPC (High Performance Computing) 環境でコンテナーを実行できます。

この例では、Open MPI から取得した ring.c プログラムに基づいています。この例では、値はリングのような方法ですべてのプロセスによって渡されます。メッセージがランク 0 に渡されるたびに、値はデクリメントされます。各プロセスが 0 メッセージを受信すると、次のプロセスに渡して終了します。0 を最初に渡すと、すべてのプロセスが 0 メッセージを取得し、通常通りに終了できます。

手順

1. Open MPI をインストールします。

```
$ sudo yum install openmpi
```

2. 環境モジュールをアクティベートするには、以下を入力します。

```
$ . /etc/profile.d/modules.sh
```

3. **mpi/openmpi-x86_64** モジュールを読み込みます。

```
$ module load mpi/openmpi-x86_64
```

必要に応じて、**mpi/openmpi-x86_64** モジュールを自動的に読み込むには、以下の行を **.bashrc** ファイルに追加します。

```
$ echo "module load mpi/openmpi-x86_64" >> .bashrc
```

4. **mpirun** と **podman** を組み合わせるには、以下の定義でコンテナーを作成します。

```
$ cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi

RUN yum -y install openmpi-devel wget && \
    yum clean all

RUN wget https://raw.githubusercontent.com/open-mpi/ompi/master/test/simple/ring.c && \
    /usr/lib64/openmpi/bin/mpicc ring.c -o /home/ring && \
    rm -f ring.c
```

5. コンテナーをビルドします。

```
$ podman build --tag=mpi-ring .
```

6. コンテナーを起動します。CPU が 4 つあるシステムでは、このコマンドはコンテナーを 4 つ起動します。

```
$ mpirun \
  --mca orte_tmpdir_base /tmp/podman-mpirun \
```

```
    podman run --env-host \
      -v /tmp/podman-mpirun:/tmp/podman-mpirun \
      --userns=keep-id \
      --net=host --pid=host --ipc=host \
      mpi-ring /home/ring
Rank 2 has cleared MPI_Init
Rank 2 has completed ring
Rank 2 has completed MPI_Barrier
Rank 3 has cleared MPI_Init
Rank 3 has completed ring
Rank 3 has completed MPI_Barrier
Rank 1 has cleared MPI_Init
Rank 1 has completed ring
Rank 1 has completed MPI_Barrier
Rank 0 has cleared MPI_Init
Rank 0 has completed ring
Rank 0 has completed MPI_Barrier
```

これにより、**mpirun** は 4 つの Podman コンテナーを開始し、各コンテナーは **ring** バイナリーのインスタンスを実行します。4 つプロセスはすべて、MPI を介して相互に通信しています。

以下の **mpirun** オプションは、コンテナーの起動に使用されます。

- **--mca orte_tmpdir_base /tmp/podman-mpirun** 行は、Open MPI に対し、**/tmp** ではなく、その一時ファイルをすべて **/tmp/podman-mpirun** に作成するように指示します。複数のノードを使用する場合、このディレクトリーは他のノードで名前が異なります。そのためには、完全な **/tmp** ディレクトリーを、より複雑なコンテナーにマウントする必要があります。

mpirun コマンドは、**podman** コマンドを起動するコマンドを指定します。以下の **podman** オプションは、コンテナーを起動するのに使用されます。

- **run** コマンドはコンテナーを実行します。
- **--env-host** オプションは、ホストからコンテナーにすべての環境変数をコピーします。
- **-v /tmp/podman-mpirun:/tmp/podman-mpirun** は、Podman に対し、Open MPI がコンテナーで利用可能な一時ディレクトリーおよびファイルを作成するディレクトリーをマウントするように指示します。
- **--userns=keep-id** 行を使用すると、コンテナー内外のユーザー ID マッピングが保証されます。
- **--net=host --pid=host --ipc=host** 行は、同じネットワーク、PID、および IPC 名前空間を設定します。
- **mpi-ring** はコンテナーの名前です。
- **/home/ring** は、コンテナー内の MPI プログラムです。

詳細は、Adwell Reber の記事「Podman in HPC environments」 を参照してください。

8.1.13. コンテナーチェックポイントの作成および復元

CRIU (Checkpoint/Restore In Userspace) は、実行中のコンテナーまたは個々のアプリケーションでチェックポイントを設定して、その状態をディスクに保存するソフトウェアです。保存されたデータを使用して、再起動後にチェックポイントが設定されたのと同じ時点でコンテナーを復元できます。

8.1.13.1. コンテナーチェックポイントのローカルでの作成および復元

この例は、リクエストの後にインクリメントされる 1 つの整数を返す Python ベースの Web サーバーに基づいています。

手順

1. Python ベースのサーバーを作成します。

```
# cat counter.py
#!/usr/bin/python3

import http.server

counter = 0

class handler(http.server.BaseHTTPRequestHandler):
    def do_GET(s):
        global counter
            s.send_response(200)
            s.send_header('Content-type', 'text/html')
            s.end_headers()
            s.wfile.write(b'%d\n' % counter)
            counter += 1


server = http.server.HTTPServer(('', 8088), handler)
server.serve_forever()
```

2. 以下の定義でコンテナーを作成します。

```
# cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi

COPY counter.py /home/counter.py

RUN useradd -ms /bin/bash counter

RUN yum -y install python3 && chmod 755 /home/counter.py

USER counter
ENTRYPOINT /home/counter.py
```

コンテナーは Universal Base Image (UBI 8) をベースとしており、Python ベースのサーバーを使用します。

3. コンテナーをビルドします。

```
# podman build . --tag counter
```

counter.py ファイルおよび **Containerfile** ファイルは、コンテナービルドプロセス (**podman build**) の入力です。ビルドされたイメージはローカルに保存され、タグ **counter** でタグ付けされます。

4. root でコンテナーを起動します。

```
# podman run --name criu-test --detach counter
```

5. 実行中のコンテナーの一覧を表示するには、次のコマンドを実行します。

```
# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED   STATUS  PORTS NAMES
e4f82fd84d48  localhost/counter:latest  5 seconds ago  Up 4 seconds ago  criu-test
```

6. コンテナーの IP アドレスを表示します。

```
# podman inspect criu-test --format "{{.NetworkSettings.IPAddress}}"
10.88.0.247
```

7. 要求をコンテナーに送信します。

```
# curl 10.88.0.247:8080
0
# curl 10.88.0.247:8080
1
```

8. コンテナーのチェックポイントを作成します。

```
# podman container checkpoint criu-test
```

9. システムを再起動します。

10. コンテナーをリストアします。

```
# podman container restore --keep criu-test
```

11. 要求をコンテナーに送信します。

```
# curl 10.88.0.247:8080
2
# curl 10.88.0.247:8080
3
# curl 10.88.0.247:8080
4
```

結果は **0** で再度開始することはありませんが、以前の値が継続されます。

これにより、再起動後に完全なコンテナーの状態を簡単に保存できます。

詳細は、Adrian Reberによる記事「Adding checkpoint/restore support to Podman」 を参照してください。

8.1.13.2. コンテナー復元を使用した起動時間の短縮

コンテナーマイグレーションを使用して、初期化に特定の時間を必要とするコンテナーの起動時間を短縮できます。チェックポイントを使用すると、同じホストまたは別のホストでコンテナーを複数回復元できます。この例は、「コンテナーチェックポイントのローカルでの作成および復元」セクションに基づいています。

手順

1. コンテナーのチェックポイントを作成し、チェックポイントイメージを **tar.gz** ファイルにエクスポートします。

```
# podman container checkpoint criu-test --export /tmp/chkpt.tar.gz
```

2. **tar.gz** ファイルからコンテナーを復元します。

```
# podman container restore --import /tmp/chkpt.tar.gz --name counter1
# podman container restore --import /tmp/chkpt.tar.gz --name counter2
# podman container restore --import /tmp/chkpt.tar.gz --name counter3
```

--name (**-n**) オプションは、エクスポートしたチェックポイントから復元されるコンテナーの新しい名前を指定します。

3. 各コンテナーの ID と名前を表示します。

```
# podman ps -a --format "{{.ID}} {{.Names}}"
a8b2e50d463c counter3
faabc5c27362 counter2
2ce648af11e5 counter1
```

4. 各コンテナーの IP アドレスを表示します。

```
#  podman inspect counter1 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.248

#  podman inspect counter2 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.249

#  podman inspect counter3 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.250
```

5. 各コンテナーに要求を送信します。

```
#  curl 10.88.0.248:8080
4
#  curl 10.88.0.249:8080
4
#  curl 10.88.0.250:8080
4
```

異なるコンテナーを使用して同じチェックポイントから復元するため、すべてのケースで結果は **4** であることに注意してください。

この方法では、最初にチェックポイントされたコンテナーのステートフルレプリカを迅速に起動できます。

詳細は、「Container migration with Podman on RHEL」を参照してください。

8.1.13.3. システム間のコンテナーの移行

この手順は、コンテナーで実行しているアプリケーションの状態を失うことなく、実行中のコンテナーをあるシステムから別のシステムに移行する方法を示しています。この例は、**counter** とタグ付けされた「コンテナーのチェックポイントの作成および復元のコンテナー」セクションに基づいています。

前提条件

以下の手順は、コンテナーがレジストリーにプッシュされている場合には不要です。Podman は、ローカルで利用できない場合にコンテナーをレジストリーから自動的にダウンロードするためです。この例ではレジストリーを使用していません。以前に構築およびタグ付けされたコンテナーをローカルにエクスポートし (「コンテナーチェックポイントのローカルでの作成および復元」セクションを参照)、この移行の宛先システムにコンテナーをインポートする必要があります。

- 以前にビルドしたコンテナーをエクスポートします。

```
# podman save --output counter.tar counter
```

- エクスポートしたコンテナーイメージを移行先システム (**other_host**) にコピーします。

```
# scp counter.tar other_host:
```

- エクスポートしたコンテナーを移行先システムにインポートします。

```
# ssh other_host podman load --input counter.tar
```

このコンテナーマイグレーションの移行先システムには、ローカルコンテナーストレージに保存されているものと同じコンテナーイメージがあります。

手順

1. root でコンテナーを起動します。

```
# podman run --name criu-test --detach counter
```

2. コンテナーの IP アドレスを表示します。

```
# podman inspect criu-test --format "{{.NetworkSettings.IPAddress}}"
10.88.0.247
```

3. 要求をコンテナーに送信します。

```
# curl 10.88.0.247:8080
0
# curl 10.88.0.247:8080
1
```

4. コンテナーのチェックポイントを作成し、チェックポイントイメージを **tar.gz** ファイルにエクスポートします。

```
# podman container checkpoint criu-test --export /tmp/chkpt.tar.gz
```

5. チェックポイントアーカイブを移行先ホストにコピーします。

```
# scp /tmp/chkpt.tar.gz other_host:/tmp/
```

6. 移行先ホスト (**other_host**) のチェックポイントを復元します。

```
# podman container restore --import /tmp/chkpt.tar.gz
```

7. 宛先ホスト (**other_host**) のコンテナーに要求を送信します。

```
# curl 10.88.0.247:8080
2
```

これにより、状態を失うことなく、あるシステムから別のシステムへステートフルコンテナーが移行されました。

詳細は、「Container migration with Podman on RHEL」 を参照してください。

8.2. RUNC

「runC」は、Open Container Initiative (OCI) コンテナーランタイム仕様の実装で、軽量で移植可能となります。runC は、コンテナーの実行を可能にする多くの低レベルの機能をまとめます。多くの低レベルコードを Docker と共有しますが、Docker プラットフォームのコンポーネントには依存しません。

runc は Linux 名前空間とライブ移行をサポートしており、移植可能なパフォーマンスプロファイルがあります。また、SELinux、コントロールグループ (cgroups)、seccomp などの Linux セキュリティー機能に完全に対応します。runc でイメージをビルドして実行したり、runc で OCI 互換イメージを実行したりできます。

8.2.1. runc でコンテナーの実行

runc では、コンテナーはバンドルを使用して設定します。コンテナーのバンドルは、「config.json」という名前の仕様ファイルと、root ファイルシステムを含むディレクトリーです。root ファイルシステムには、コンテナーの内容が含まれます。

バンドルを作成するには、以下を実行します。

```
$ runc spec
```

このコマンドは、編集が必要な、必要最小限の構造のみを含む config.json ファイルを作成します。最も重要なことは、実行するファイルを指定するために「args」パラメーターを変更する必要があるということです。デフォルトでは、「args」は「sh」に設定されています。

```
 "args": [
   "sh"
 ],
```

たとえば、podman を使用して Red Hat Enterprise Linux ベースイメージ (ubi8/ubi) をダウンロードしてエクスポートし、runc でそのイメージ用の新しいバンドルを作成し、「config.json」ファイルを編集してそのイメージを指定します。次に、コンテナーイメージを作成し、runc を使用してそのイメージのインスタンスを実行します。以下のコマンドを使用します。

```
# podman pull registry.redhat.io/ubi8/ubi
```

```
# podman export $(podman create registry.redhat.io/ubi8/ubi) > rhel.tar
# mkdir -p rhel-runc/rootfs
# tar -C rhel-runc/rootfs -xf rhel.tar
# runc spec -b rhel-runc
# vi rhel-runc/config.json   Change any setting you like
# runc create -b rhel-runc/ rhel-container
# runc start rhel-container
sh-4.2#
```

この例では、コンテナーインスタンスの名前は「rhel-container」です。そのコンテナーを実行すると、デフォルトでシェルを起動するため、そのコンテナーからコマンドの検索や実行が可能になります。終了したら、**exit** と入力します。

コンテナーインスタンスの名前は、ホストで一意のものである必要があります。コンテナーの新しいインスタンスを起動するには、以下を実行します。

```
# runc start <container_name>
```

bundle ディレクトリーは、「-b」オプションで提供できます。デフォルトでは、bundle の値は現在のディレクトリーになります。

runc でコンテナーを起動するには、root 権限が必要になります。runc で利用可能なコマンドとその使用方法をすべて表示するには、「runc --help」を実行します。

8.3. SKOPEO

skopeo コマンドを使用すれば、docker デーモンや **docker** コマンドを使用しなくても、レジストリーからコンテナーイメージを扱うことができます。レジストリーには、Docker レジストリー、独自のローカルレジストリー、Red Hat Quay、または OpenShift レジストリーを追加できます。skopeo で実行できることは次のとおりです。

- **検証** - **skopeo inspect** コマンドの出力は、**docker inspect** コマンドで表示されるもの (コンテナーイメージに関する詳細な情報) と似ています。この出力は、json 形式 (デフォルト) または raw 形式 (--raw オプションを使用) の可能性があります。
- **コピー** - **skopeo copy** を使用すると、コンテナーイメージをレジストリーから別のレジストリーまたはローカルディレクトリーにコピーできます。
- **レイヤー** - **skopeo layers** コマンドを使用すると、イメージに関連付けられているレイヤーをダウンロードして、tarball および関連付けられているマニフェストファイルとしてローカルディレクトリーに保存できます。

コンテナーまたはイメージライブラリーに依存する **buildah** コマンドやその他のツールのように、**skopeo** コマンドは、Docker に関連付けられているもの以外のコンテナーストレージ領域からのイメージを処理できます。他のタイプのコンテナーストレージへの利用可能な変換には、コンテナーストレージ (**buildah** および CRI-O が保存するイメージ用)、ostree (アトミックおよびシステムコンテナー用)、oci (OCI 準拠のディレクトリーに格納されているコンテンツ用) などがあります。詳細は skopeo の man ページ を参照してください。

skopeo を試すには、ローカルレジストリーを設定してから、イメージレイヤーの検証、コピー、およびダウンロードを行うコマンドを実行します。この例に従う場合は、まず次の手順を実行します。

- ローカルレジストリー(Red Hat Quay など) をインストールします。RHEL 7 の docker-distribution パッケージで利用可能なコンテナーレジストリーソフトウェアは、RHEL 8 では利用できません。

- 最新の RHEL イメージをローカルシステム (**podman pull ubi8/ubi**) にプルします。
- RHEL イメージのタグを再登録し、以下のようにローカルレジストリーにプッシュします。

```
# podman tag ubi8/ubi localhost/myubi8
# podman push localhost/myubi8
```

本セクションの残りの部分では、RHEL イメージからレイヤーを検証、コピー、および取得する方法を説明します。

注記

デフォルトでは、**skopeo** ツールは TLS 接続を必要とします。暗号化されていない接続を使用しようとすると失敗します。デフォルトを上書きして http レジストリーを使用するには、文字列 **<registry>/<image>** の前に **http:** と付けます。

8.3.1. skopeo でコンテナーイメージの検証

レジストリーからコンテナーイメージを検証する場合は、コンテナーの形式 (docker など)、レジストリーの場所 (docker.io、localhost など)、およびリポジトリー/イメージ (ubi8/ubi など) を指定する必要があります。

以下の例では、Docker レジストリーから mariadb コンテナーイメージを検証します。

```
# skopeo inspect docker://docker.io/library/mariadb
{
    "Name": "docker.io/library/mariadb",
    "Tag": "latest",
    "Digest": "sha256:d3f56b143b62690b400ef42e876e628eb5e488d2d0d2a35d6438a4aa841d89c4",
    "RepoTags": [
        "10.0.15",
        "10.0.16",
        "10.0.17",
        "10.0.19",
...
    "Created": "2018-06-10T01:53:48.812217692Z",
    "DockerVersion": "1.10.3",
    "Labels": {},
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
...
```

タグ付けされた **localhost/myubi8** を、ローカルシステムで実行しているコンテナーリポジトリーへプッシュしたとすると、次のコマンドでそのイメージを検証します。

```
# skopeo inspect docker://localhost/myubi8
{
    "Name": "localhost/myubi8",
    "Tag": "latest",
    "Digest": "sha256:4e09c308a9ddf56c0ff6e321d135136eb04152456f73786a16166ce7cba7c904",
    "RepoTags": [
        "latest"
    ],
```

```
    "Created": "2018-06-16T17:27:13Z",
    "DockerVersion": "1.7.0",
    "Labels": {
        "Architecture": "x86_64",
        "Authoritative_Registry": "registry.access.redhat.com",
        "BZComponent": "rhel-server-docker",
        "Build_Host": "rcm-img01.build.eng.bos.redhat.com",
        "Name": "myubi8",
        "Release": "75",
        "Vendor": "Red Hat, Inc.",
        "Version": "8.0"
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf"
    ]
}
```

8.3.2. skopeo でコンテナーイメージのコピー

このコマンドは、myubi8 コンテナーイメージを、ローカルレジストリーからローカルシステムのディレクトリーにコピーします。

```
# skopeo copy docker://localhost/myubi8 dir:/root/test/
INFO[0000] Downloading
myubi8/blobs/sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf
# ls /root/test
16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf.tar manifest.json
```

skopeo copy コマンドの結果は、指定したディレクトリーへコピーするイメージを表す tarball (16d*.tar) と manifest.json です。レイヤーが複数になる場合は、tarball も複数になります。**skopeo copy** コマンドは、別のレジストリーにイメージをコピーすることもできます。宛先レジストリーに書き込むのに使用する署名を提供する必要がある場合は、コマンドラインに **--sign-by=** オプションを追加し、その後に必要な key-id を追加します。

8.3.3. skopeo でイメージレイヤーの取得

skopeo layers コマンドは、**skopeo copy** に似ており、相違点は、**copy** オプションはイメージを別のレジストリーまたはローカルディレクトリーにコピーできますが、**layers** オプションは現在のディレクトリーにあるレイヤー (tarball ファイルおよび manifest.json ファイル) を削除するだけとなります。以下に例を示します。

```
# skopeo layers docker://localhost/myubi8
INFO[0000] Downloading
myubi8/blobs/sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf
# find .
./layers-myubi8-latest-698503105
./layers-myubi-latest-698503105/manifest.json
./layers-myubi8-latest-
698503105/16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf.tar
```

この例では、新しいディレクトリー (layers-myubi8-latest-698503105) を作成し、このケースでは 1 つのレイヤーの tarball と manifest.json ファイルがそのディレクトリーにコピーされます。

第9章 関連情報

- Buildah - OCI コンテナーイメージを構築するツール
- Podman - コンテナーを実行および管理するツール
- Skopeo - コンテナーイメージのコピーと検証を行うツール