Red Hat Enterprise Linux 9

コンテナーの構築、実行、および管理

Red Hat Enterprise Linux 9 での Linux コンテナーの構築、実行、および管理

Last Updated: 2022-08-15

Red Hat Enterprise Linux 9 コンテナーの構築、実行、および管理

Red Hat Enterprise Linux 9 での Linux コンテナーの構築、実行、および管理

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本書は、podman、buildah、skopeo、runc、crun などのコマンドラインツールを使用して、Red Hat Enterprise Linux 9 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

多様性を受け入れるオープンソースの強化 6

RED HAT ドキュメントへのフィードバックの提供 7

第1章 コンテナーの使用 8
1.1. PODMAN、BUILDAH、および SKOPEO の特徴 8
1.2. PODMAN コマンドの概要 9
1.3. DOCKER を使用せずにコンテナーを実行 11
1.4. コンテナーの RHEL アーキテクチャーの選択 11
1.5. コンテナーツールの取得 12
1.6. ルートレスコンテナーの設定 13
1.7. ルートレスコンテナーへのアップグレード 14
1.8. ルートレスコンテナーに関する特別な考慮事項 15
1.9. 関連情報 16

第2章 コンテナーイメージの種類 17
2.1. RHEL コンテナーイメージの一般的な特徴 17
2.2. UBI イメージの特徴 17
2.3. UBI 標準イメージの概要 18
2.4. UBI INIT イメージの概要 19
2.5. UBI の最小イメージの理解 19
2.6. UBI マイクロイメージの概要 20
2.7. UBI INIT イメージの使用 20
2.8. UBI マイクロイメージの使用 21

第3章 コンテナーイメージの使用 23
3.1. コンテナーレジストリー 23
3.2. コンテナーレジストリーの設定 24
3.3. コンテナーイメージの検索 25
3.4. レジストリーからのイメージの取得 (プル) 26
3.5. 短縮名のエイリアスの設定 26
3.6. 短縮名のエイリアスを使用したコンテナーイメージのプル 27
3.7. イメージの一覧表示 28
3.8. ローカルイメージの検証 29
3.9. リモートイメージの検証 29
3.10. コンテナーイメージのコピー 30
3.11. ローカルディレクトリーへのイメージレイヤーのコピー 30
3.12. タグ付けイメージ 31
3.13. イメージの保存および読み込み 32
3.14. UBI イメージの再配布 33
3.15. イメージ署名のデフォルト検証 34
3.16. イメージの削除 35

第4章 コンテナーの使用 37
4.1. PODMAN RUN コマンド 37
4.2. ホストからのコンテナーでのコマンド実行 37
4.3. コンテナー内でのコマンドの実行 38
4.4. コンテナーの一覧表示 39
4.5. コンテナーの起動 40
4.6. ホストからのコンテナーの検証 40
4.7. LOCALHOST のディレクトリーのコンテナーへのマウント 42
4.8. コンテナーのファイルシステムのマウント 42

4.9. 静的 IP を使用したデーモンとしてのサービスの実行 43
4.10. 実行中のコンテナー内でのコマンドの実行 44
4.11. 2 つのコンテナー間でのファイルの共有 45
4.12. コンテナーのエクスポートおよびインポート 47
4.13. コンテナーの停止 49
4.14. コンテナーの削除 50
4.15. RUNC コンテナーランタイム 51
4.16. CRUN コンテナーランタイム 51
4.17. RUNC および CRUN でのコンテナーの実行 52
4.18. コンテナーランタイムの一時的な変更 53
4.19. コンテナーランタイムの永続的な変更 54
4.20. コンテナーの SELINUX ポリシーの作成 55

第5章 POD の使用 56
5.1. POD の作成 56
5.2. POD 情報の表示 57
5.3. POD の停止 59
5.4. POD の削除 59

第6章 コンテナーネットワークの管理 61
6.1. コンテナーネットワークのリストアップ 61
6.2. ネットワークの検査 61
6.3. ネットワークの作成 62
6.4. コンテナーのネットワークへの接続 63
6.5. コンテナーのネットワークからの切断 63
6.6. ネットワークの削除 64
6.7. 未使用の全ネットワークの削除 65

第7章 コンテナー間の通信 66
7.1. ネットワークモードとレイヤー 66
7.2. コンテナーのネットワーク設定の検査 66
7.3. コンテナーとアプリケーション間の通信 66
7.4. コンテナーとホスト間の通信 67
7.5. ポートマッピングによるコンテナー間の通信 68
7.6. DNS を利用したコンテナー間の通信 69
7.7. POD 内の 2 つのコンテナー間での通信 70
7.8. POD での通信 71
7.9. コンテナーネットワークへの POD のアタッチ 71

第8章 コンテナーネットワークモードの設定 73
8.1. 静的 IP でのコンテナー実行 73
8.2. SYSTEMD なしでの DHCP プラグイン実行 73
8.3. SYSTEMD を使用した DHCP プラグインの実行 74
8.4. MACVLAN プラグイン 75
8.5. ネットワークスタックの CNI から NETAVARK への切り替え 76
8.6. NETAVARK から CNI へのネットワークスタックの切り替え 77

第9章 PODMAN を使用した OPENSHIFT へのコンテナーの移植 79
9.1. PODMAN を使用した KUBERNETES YAML ファイルの生成 79
9.2. OPENSHIFT 環境での KUBERNETES YAML ファイルの生成 80
9.3. PODMAN でのコンテナーおよび POD の起動 81
9.4. OPENSHIFT 環境でのコンテナーおよび POD の起動 81
9.5. PODMAN を使用したコンテナーと POD の手動実行 82
9.6. PODMAN を使用した YAML ファイルの生成 84

9.7. PODMAN を使用したコンテナーと POD の自動実行 86
9.8. PODMAN を使用した POD の自動停止/削除 87

第10章 PODMAN を使用した SYSTEMD へのコンテナーの移植 89
10.1. SYSTEMD サービスの有効化 89
10.2. PODMAN を使用した SYSTEMD ユニットファイルの生成 90
10.3. PODMAN を使用した SYSTEMD ユニットファイルの自動生成 91
10.4. SYSTEMD を使用したコンテナーの自動起動 93
10.5. SYSTEMD を使用した POD の自動起動 95
10.6. PODMAN を使用したコンテナーの自動更新 98
10.7. SYSTEMD を使用したコンテナーの自動更新 99

第11章 実行中の UBI コンテナーへのソフトウェアの追加 101
11.1. 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) 101
11.2. 標準の UBI コンテナーへのソフトウェアの追加 101
11.3. 最小 UBI コンテナーへのソフトウェアの追加 102
11.4. 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) 103
11.5. UBI ベースのイメージの構築 104
11.6. アプリケーションストリームランタイムイメージの使用 105
11.7. UBI コンテナーイメージソースコードの取得 105

第12章 コンテナーでの SKOPEO、BUILDAH、および PODMAN の実行 108
12.1. コンテナーでの SKOPEO の実行 108
12.2. 認証情報を使用したコンテナーでの SKOPEO の実行 110
12.3. AUTHFILE を使用したコンテナーでの SKOPEO の実行 110
12.4. ホストに対するコンテナーイメージのコピー 111
12.5. コンテナーでの BUILDAH の実行 112
12.6. 特権および非特権 PODMAN コンテナー 113
12.7. 拡張された権限での PODMAN の実行 113
12.8. 少ない権限での PODMAN の実行 114
12.9. PODMAN コンテナー内でのコンテナーのビルド 115

第13章 BUILDAH でコンテナーイメージの構築 118
13.1. BUILDAH ツール 118
13.2. BUILDAH のインストール 118
13.3. BUILDAH でイメージの取得 119
13.4. コンテナー内でのコマンドの実行 120
13.5. BUILDAH を使用した CONTAINERFILE からのイメージのビルド 120
13.6. BUILDAH でコンテナーおよびイメージの検証 121
13.7. BUILDAH MOUNT を使用したコンテナーの変更 122
13.8. BUILDAH COPY および BUILDAH CONFIG を使用したコンテナーの変更 123
13.9. BUILDAH でゼロからイメージを新規作成 125
13.10. プライベートレジストリーへのコンテナーのプッシュ 126
13.11. DOCKER HUB へのコンテナーのプッシュ 127
13.12. BUILDAH でイメージの削除 128
13.13. BUILDAH でコンテナーの削除 129

第14章 コンテナーの監視 130
14.1. コンテナーでの HEALTHCHECK の実行 130
14.2. PODMAN システム情報の表示 131
14.3. PODMAN イベントタイプ 134
14.4. PODMAN イベントのモニタリング 136

第15章 コンテナーチェックポイントの作成および復元 138
15.1. ローカルでのコンテナーチェックポイントの作成および復元 138

15.2. コンテナー復元を使用した起動時間の短縮 140
15.3. システム間のコンテナーの移行 141

第16章 HPC 環境での PODMAN の使用 143
16.1. PODMAN とMPI の使用 143
16.2. MPIRUN オプション 144

第17章 特殊なコンテナーイメージの実行 146
17.1. ホストへの権限の付与 146
17.2. RUNLABEL が組み込まれたコンテナーイメージ 146
17.3. ランレベルでの RSYSLOG の実行 146

第18章 CONTAINER-TOOLS API の使用 149
18.1. ROOT モードで SYSTEMD を使用した PODMAN API の有効化 149
18.2. ルートレスモードで SYSTEMD を使用した PODMAN API の有効化 150
18.3. PODMAN API の手動実行 150

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社の CTO、Chris Wright のメッセージ を参照してください。

RED HAT ドキュメントへのフィードバックの提供

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。

- 特定の部分についての簡単なコメントをお寄せいただく場合は、以下をご確認ください。
    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隅に **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。
- Bugzilla を介してフィードバックを送信するには、新しいチケットを作成します。
    1. Bugzilla の Web サイトに移動します。
    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。RHEL は、以下のようなコア技術を使用して Linux コンテナーを実装します。

- 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 namespace
- SELinux (セキュリティー用)
- セキュアなマルチテナンシー

このような技術は、セキュリティーエクスプロイトの可能性を軽減し、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します。

Red Hat OpenShift は、強力なコマンドラインと Web UI ツールを提供し、Pod と呼ばれる単位でコンテナーを構築、管理、および実行します。Red Hat では、OpenShift 外で個々のコンテナーおよびコンテナーイメージをビルドして管理できます。本書では、RHEL システムで直接実行されるタスクを実行するためのツールについて説明します。

他のコンテナー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、Red Hat は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 およびコンテナーイメージの直接管理 (**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 へのコンテナーの実行機能と構築機能の提供
- **crun** - ルートレスコンテナーの柔軟性、制御、セキュリティーを向上するために設定可能なオプションのランタイム。

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。

マルチノードコンテナープラットフォームの場合は、「OpenShift」および「CRI-O Container Engine の使用」を参照してください。

1.1. PODMAN、BUILDAH、および SKOPEO の特徴

Docker コマンド機能に代わる、Podman、Skopeo、Buildah 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を当てています。

Podman、Skopeo、Buildah ツールの主な利点は次のとおりです。

- ルートレスモードでの実行 - ルートレスコンテナーは、特権を追加しなくても実行されるため、はるかに安全です。

- デーモンの必要なし - コンテナーが実行されていない場合、Podmanは実行されないので、これらのツールではアイドル時のリソース要件がはるかに少なくなります。一方、docker は常にデーモンを実行します。
- ネイティブ systemd 統合 - Podman では systemd ユニットファイルを作成し、コンテナーをシステムサービスとして実行できます。

Podman、Skopeo、および Buildah の特徴は次のとおりです。

- 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。このツールはイメージを共有できます。
- プログラムで Podman と対話するには、ルートおよびルートレス環境の両方で動作する Podman v2.0 RESTful API を使用してください。詳細は、「コンテナーツール API の使用」 の章を参照してください。

関連情報

- Say "Hello" to Buildah, Podman, and Skopeo
- Podman and Buildah for Docker users
- 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

1.2. PODMAN コマンドの概要

表 1.1 は、**podman** コマンドで使用できるコマンドの一覧です。**podman -h** を使用して、すべての Podman コマンドの一覧を表示します。

表1.1 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	Containerfile 命令を使用してイメージをビルドする	**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		

関連情報

- Podman Basics Cheat Sheet
- 5 Podman features to try now

1.3. DOCKER を使用せずにコンテナーを実行

Red Hat では、RHEL 9 から Docker コンテナーエンジンと、docker コマンドが削除されました。

RHEL で Docker を使用する場合は、異なるアップストリームプロジェクトから Docker を取得できますが、RHEL 9 では対応していません。

- **podman-docker** パッケージをインストールできます。**docker** コマンドを実行するたびに、実際には**podman**コマンドが実行されます。
- Podman は Docker Socket API にも対応しているため、**podman-docker** パッケージは **/var/run/docker.sock** と **/var/run/podman/podman.sock** の間でリンクを設定します。そのため、Docker デーモンを必要とせずに、**docker-py** と **docker-compose** ツールを使用して Docker API コマンドをそのまま実行できます。Podman はこのような要求を処理します。
- **docker** コマンドなどの **podman** コマンドは、**Containerfile** または **Dockerfile** からコンテナーイメージを構築できます。**Containerfile** および **Dockerfile** 内で使用できる利用可能なコマンドは同じです。
- **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。

関連情報

- Podman and Buildah for Docker users

1.4. 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)
- 64 ビット IBM Z (ベースイメージと、大概の階層構造イメージ)
- ARM 64 ビット (ベースイメージのみ)

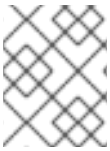
初めは、すべてのアーキテクチャーですべての Red Hat イメージがサポートされたわけではありませんが、一覧に挙げられているすべてのアーキテクチャーでほぼすべてが利用可能になりました。

関連情報

- Universal Base Images (UBI): イメージ、リポジトリー、およびパッケージ

1.5. コンテナーツールの取得

この手順では、Podman、Buildah、Skopeo、CRIU、Udica といった必要なライブラリーを含む **container-tools** メタパッケージをインストールする方法を説明します。



注記

安定したストリームは RHEL 9 では利用できません。Podman、Buildah、Skopeo などへの安定したアクセスを受けるには、RHEL EUS サブスクリプションを使用します。

手順

1. RHEL をインストールします。
2. RHEL の登録:ユーザー名とパスワードを入力します。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す。

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

3. RHEL にサブスクライブします。
   - RHEL に自動的にサブスクライブするには、以下を実行します。

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

   - プール ID で RHEL をサブスクライブするには、次のコマンドを実行します。

```
# subscription-manager attach --pool PoolID
```

4. **container-tools** メタパッケージをインストールします。

```
# dnf install container-tools
```

5. オプション。**podman-docker** パッケージをインストールします。

```
# dnf install -y podman-docker
```

podman-docker パッケージは、Docker コマンドラインインターフェースと **docker-api** を、同等の Podman コマンドに置き換えます。

1.6. ルートレスコンテナーの設定

システムで利用可能な機能をコンテナーで完全利用できるように、Podman、Skopeo、Buildah などのコンテナーツールをスーパーユーザー特権 (root ユーザー) が割り当てられたユーザーで実行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。これにより、一般ユーザーがコンテナー経由で要求を送信でき、システムに影響を与える可能性があります。システム管理者は、ルートレスコンテナーユーザーを設定して、一般ユーザーがコンテナーアクティビティーに損害を与える可能性を回避しつつ、一般ユーザーが自分のアカウントで大半のコンテナー機能を安全に実行できるようにします。

この手順では、Podman、Skopeo、および Buildah ツールを使用して、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システムを設定する方法を説明します。また、一般ユーザーのアカウントは、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に発生する可能性のある制限についても説明します。

前提条件

- 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。

手順

1. RHEL をインストールします。
2. **podman** パッケージをインストールします。

```
# dnf install podman -y
```

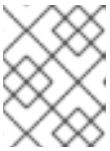
3. ユーザーアカウントを新規作成します。

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

   - ユーザーは、ルートレス Podman を使用できるように自動的に設定されます。
   - **useradd** コマンドは、**/etc/subuid** と **/etc/subgid** ファイルに、アクセス可能なユーザー ID とグループ ID の範囲を自動的に設定します。
   - **etc/subuid** や **/etc/subgid** を手動で変更した場合、新しい変更を適用させるために **podman system migrate** コマンドを実行する必要があります。

4. ユーザーに接続します。

```
$ ssh joe@server.example.com
```



注記

これらのコマンドでは正しい環境変数が設定されないため、**su** または **su -** コマンドは使用しないでください。

5. **registry.access.redhat.com/ubi9/ubi** コンテナーイメージをプルします。

```
$ podman pull registry.access.redhat.com/ubi9/ubi
```

6. **myubi** という名前のコンテナーを実行し、OS バージョンを表示します。

```
$ podman run --rm --name=myubi registry.access.redhat.com/ubi9/ubi cat \
    /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="9 (Plow)"
```

関連情報

- Podman を使用したルートレスコンテナー: 基本的なコンテナー
- **podman-system-migrate** の man ページ

1.7. ルートレスコンテナーへのアップグレード

本セクションでは、RHEL 7 からルートレスコンテナーにアップグレードする方法を説明します。ユーザーおよびグループ ID を手動で設定する必要があります。

以下は、RHEL 7 からルートレスコンテナーにアップグレードするお t 機に考慮すべき事項です。

- 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 に 65536 を使用します。ただし、この数は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)。

前提条件

- ユーザーアカウントが作成されている。

手順

- **usermod** コマンドを実行して、UID と GID をユーザーに割り当てます。

```
# usermod --add-subuids 200000-201000 --add-subgids 200000-201000 username
```

  - **usermod --add-subuid** コマンドは、アクセス可能なユーザー ID の範囲をユーザーのアカウントに追加します。

- **usermod --add-subgids** コマンドは、アクセス可能なユーザー GID およびグループ ID の範囲をユーザーのアカウントに手動で追加します。

検証手順

- UID および GID が正しく設定されていることを確認します。

```
# grep username /etc/subuid /etc/subgid
#/etc/subuid:username:200000:1001
#/etc/subgid:username:200000:1001
```

1.8. ルートレスコンテナーに関する特別な考慮事項

root 以外のユーザーでコンテナーを実行する場合は、考慮事項が複数あります。

- ホストコンテナーストレージへのパスは、root ユーザー (**/var/lib/containers/storage**) と root 以外のユーザー (**$HOME/.local/share/containers/storage**) との間では異なります。

- 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、ホストのオペレーティングシステムに対する root 権限はありません。

- **etc/subuid** や **/etc/subgid** を手動で変更した場合、新しい変更を適用させるために **podman system migrate** コマンドを実行する必要があります。

- 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に設定ファイルを作成します (**$HOME/.config/containers**)。設定ファイルには、**storage.conf** (ストレージ設定用) および **containers.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。

- root 権限なしで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**) を実行してシステムクロックを変更できません。root としてコンテナーを実行し、ルートレスコンテナー環境を省略して root ユーザーの環境を使用する必要があります。以下は例になります。

```
$ 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を調整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 未満のポート番号にアクセスできません。たとえば、ルートレスコンテナーの namespace 内では、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、この namespace の外部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 未満のポートでサービスを公開できるようにしますが、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、外部ユーザーに対して、管理者が設定したと見せかけることができます。これは、テスト用のワークステーションで問題ありませんが、ネットワークにアクセス可能な開発サーバーでは適切ではなく、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
```

関連情報

- ルートレス Podman の欠点

1.9. 関連情報

- A Practical Introduction to Container Terminology

第2章 コンテナーイメージの種類

コンテナーイメージは、単一コンテナー実行の全要件、およびそのニーズおよび機能を記述するメタデータを含むバイナリーです。

コンテナーイメージには、以下の 2 つのタイプがあります。

- Red Hat Enterprise Linux Base Images (RHEL ベースイメージ)
- Red Hat Universal Base Images (UBI イメージ)

どちらのタイプのコンテナーイメージも Red Hat Enterprise Linux の一部から構築されます。これらのコンテナーを使用することで、ユーザーは、信頼性、セキュリティー、パフォーマンス、ライフサイクルを最大限に活用できます。

2 種類のコンテナーイメージの主な違いは、UBI イメージではコンテナーイメージを他のタイプと共有できる点です。UBI を使用してコンテナー化アプリケーションをビルドして、任意のレジストリーサーバーにプッシュし、他のサーバーと簡単に共有し、Red Hat 以外のプラットフォームにもデプロイできます。UBI イメージは、コンテナーで開発されるクラウドネイティブおよび Web アプリケーションユースケースの基礎として設計されています。

2.1. RHEL コンテナーイメージの一般的な特徴

以下の特徴は、RHEL ベースイメージと UBI イメージの両方に当てはまります。

一般的には、RHEL コンテナーイメージの特徴は以下のとおりです。

- **サポート対象**:Red Hat では、コンテナー化されたアプリケーションでの使用をサポートしています。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**:「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**:明確に定義された更新スケジュールで提供されます。最新のソフトウェアを取得するには、Red Hat Container Image の更新記事 を参照してください。
- **追跡**:Red Hat 製品エラータにより追跡され、各更新に追加された変更を理解するのに役立ちます。
- **再利用可能**:コンテナーイメージは、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。各コンテナーイメージは、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。

2.2. UBI イメージの特徴

UBI イメージを使用すると、コンテナーイメージを他と共有できます。4 つの UBI イメージ (Micro、Minimal、Standard、および init) が提供されます。アプリケーションのビルド用に、事前にビルドされた言語のランタイムイメージと DNF リポジトリーが提供されます。

UBI イメージの特徴は以下のとおりです。

- **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。

- **再配布可能**:UBI イメージは、Red Hat のお客様、パートナー、ISV など向けに標準化できます。UBI イメージを使用すると、自由に共有およびデプロイが可能な公式の Red Hat ソフトウェアにコンテナーイメージを構築できます。
- **4 つのベースイメージをセットで提供する**: Micro、Minimal、Standard、init
- **ビルド済みの言語ランタイムコンテナーイメージのセットを提供します。**Application Streams をベースとするランタイムイメージは、アプリケーションの基盤を提供し、python、perl、php、dotnet、nodejs、ruby などの標準かつサポート対象のランタイムから利点を受けることができます。
- **関連のある DNF リポジトリーを提供する**: DNF リポジトリーには、RPM パッケージと更新が含まれており、アプリケーションの依存関係を追加して、UBI コンテナーイメージを再ビルドできます。
  - **ubi-9-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-9-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アプリケーションストリームパッケージを保持しています。
  - **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。
- **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。

関連情報

- Red Hat Universal Base Image の概要
- Universal Base Images (UBI): イメージ、リポジトリー、およびパッケージ
- Red Hat Universal Base Image に必要な情報
- よくある質問 - Universal Base Image

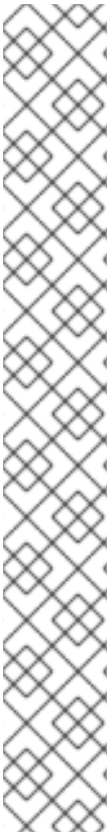
2.3. UBI 標準イメージの概要

標準イメージ (名称 **ubi**) は、RHEL で実行されるアプリケーション用に設計されています。UBI 標準イメージの主な機能には、以下が含まれます。

- **init システム**: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。
- **dnf**: ソフトウェアの追加や更新が可能な、無料の dnf リポジトリーにアクセスできます。**dnf** コマンドの標準セット（**dnf**、**dnf-config-manager**、**dnfdownloader** など）を使用できます。
- **ユーティリティー**:ユーティリティには **tar**、**dmidecode**、**gzip**、**getfacl** や他の ACL コマンド、**dmsetup**、ここに記載されていない他のユーティリティーとのデバイスマッパーコマンドが含まれます。

2.4. UBI INIT イメージの概要

UBI init イメージ (名称 **ubi8-init**) 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

ubi9-init イメージは **ubi9** 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。

- **ubi9-init**:
  - CMD は **/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。
  - **ps** およびプロセス関連のコマンド (**procps-ng** パッケージ) が含まれます。
  - また、**SIGRTMIN+3** を **StopSignal** として設定しています。これは、**ubi9-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。
- **ubi9**:
  - CMD は **/bin/bash** に設定されます。
  - **ps** およびプロセス関連のコマンド (**procps-ng** パッケージ) は含まれません。
  - 通常の終了シグナルを無視しません (**SIGTERM** および **SIGKILL**)。

2.5. UBI の最小イメージの理解

ubi-minimal という名前の UBI の最小イメージは、事前にインストールされたコンテンツセットおよびパッケージマネージャー (microdnf) の最小イメージを提供します。これにより、**Containerfile** を使用し、イメージに含まれる依存関係を最小化できます。

UBI 最小イメージの主な機能には、以下が含まれます。

- **サイズが小さい**: 最小イメージは、ディスク上では約 92M、圧縮時は 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に開発された **dnf** 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あるため、リポジトリーの有効化/無効化、パッケージの削除と更新、パッケージのインストール後にキャッシュを削除できます。
- **RHEL パッケージをベースとする**: 最小イメージには、通常の RHEL ソフトウェアの RPM パッケージから機能がいくつか削除されたものです。最小イメージには、systemd や System V init、Python ランタイム環境、および一部のシェルユーティリティーなど、初期化およびサービス管理システムが含まれません。RHEL リポジトリーを使用してイメージを構築することができますが、オーバーヘッドの量は最小限に抑えられます。
- **microdnfのモジュールがサポートされている**: **microdnf** 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。**microdnf module enable**、**microdnf module disable**、および **microdnf module reset** を使用して、モジュールストリームをそれぞれ有効化、無効化、およびリセットできます。

- たとえば、UBI 最小コンテナー内で **nodejs:14** モジュールストリームを有効にするには、次のコマンドを実行します。

```
# microdnf module enable nodejs:14
Downloading metadata...
...
Enabling module streams:
   nodejs:14

Running transaction test...
```

Red Hat は UBI の最新バージョンのみをサポートし、ドットリリースの過去バージョンはサポートしません。特定のドットリリースを使用し続ける必要がある場合は、「延長アップデートサポート」を参照してください。

2.6. UBI マイクロイメージの概要

ubi-micro は、取得可能な最小 UBI イメージで、パッケージマネージャーと、通常はコンテナーイメージに含まれるすべての依存関係を除外しています。これにより、他のアプリケーションに UBI Standard、Minimal、または Init を使用する場合でも、**ubi-micro** イメージがベースのコンテナーイメージに対する攻撃の可能性を最小限に抑えられ、最小アプリケーションに適しています。Linux ディストリビューションパッケージのないコンテナーイメージは Distroless コンテナーイメージと呼ばれます。

2.7. UBI INIT イメージの使用

この手順では、**Containerfile** を使用してコンテナーを構築する方法を紹介します。この Containerfile は、コンテナーがホストシステムで実行されている場合に Web サーバー (**httpd**) をインストールして、systemd サービス (**/sbin/init**) により Web サーバーが自動的に起動されるように設定します。**podman build** コマンドは、1 つ以上の **Containerfiles** および指定されたビルドコンテキストディレクトリーにある命令を使用してイメージをビルドします。コンテキストディレクトリーは、アーカイブ、Git リポジトリー、または **Containerfile** の URL として指定できます。コンテキストディレクトリーが指定されていない場合、現在の作業ディレクトリーはビルドコンテキストと見なされ、**Containerfile** が含まれている必要があります。**--file** オプションで **Containerfile** を指定することもできます。

手順

1. 新しいディレクトリーに以下の内容を含む **Containerfile** を作成します。

```
FROM registry.access.redhat.com/ubi9/ubi-init
RUN dnf -y install httpd; dnf 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
CMD [ "/sbin/init" ]
```

Containerfile 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. コンテナーをビルドします。

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

3. オプション。お使いのシステムで systemd を使用してコンテナーを実行し、SELinux を有効にする場合は、**container_manage_cgroup** ブール値変数を設定する必要があります。

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

4. **mysysd_run** という名前のコンテナーを実行します。

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

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。

注記

ルートレスモードでは、1024 以上のホストのポート番号を選択する必要があります。以下は例になります。

```
$ podman run -d --name=mysysd -p 8081:80 mysysd
```

1024 未満のポート番号を使用するには、**net.ipv4.ip_unprivileged_port_start** 変数を変更する必要があります。

```
$ sudo sysctl net.ipv4.ip_unprivileged_port_start=80
```

5. コンテナーが実行していることを確認します。

```
# podman ps
a282b0c2ad3d  localhost/mysysd:latest  /sbin/init  15 seconds ago  Up 14 seconds ago
0.0.0.0:80->80/tcp  mysysd_run
```

6. Web サーバーをテストします。

```
# curl localhost/index.html
Successful Web Server Test
```

関連情報

- ルートレス Podman の欠点

2.8. UBI マイクロイメージの使用

この手順では、Buildah ツールを使用して **ubi-micro** コンテナーイメージを構築する方法を説明します。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

1. **registry.access.redhat.com/ubi8/ubi-micro** イメージをプルしてビルドします。

```
# microcontainer=$(buildah from registry.access.redhat.com/ubi9-beta/ubi-micro)
```

2. 作業中のコンテナーの root ファイルシステムをマウントします。

```
# micromount=$(buildah mount $microcontainer)
```

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

```
# dnf install \
    --installroot $micromount \
    --releasever=/ \
    --setopt install_weak_deps=false \
    --setopt=reposdir=/etc/yum.repos.d/ \
    --nodocs -y \
    httpd
# dnf clean all \
    --installroot $micromount
```

4. 作業コンテナーで root ファイルシステムをアンマウントします。

```
# buildah umount $microcontainer
```

5. 作業コンテナーから **ubi-micro-httpd** イメージを作成します。

```
# buildah commit $microcontainer ubi-micro-httpd
```

検証手順

1. **ubi-micro-httpd** イメージの詳細を表示します。

```
# podman images ubi-micro-httpd
localhost/ubi-micro-httpd latest 7c557e7fbe9f  22 minutes ago  151 MB
```

第3章 コンテナーイメージの使用

Podman ツールは、コンテナーイメージと連携するように設計されています。このツールを使用して、イメージのプル、イメージ署名の検査、タグ付け、保存、読み込み、再配布、および定義を行うことができます。

3.1. コンテナーレジストリー

コンテナーレジストリーは、コンテナーイメージとコンテナーベースのアプリケーションアーティファクトを保存するためのリポジトリーまたはリポジトリーのコレクションです。Red Hat が提供するレジストリーは以下のとおりです。

- registry.redhat.io (認証が必要)
- registry.access.redhat.com (認証なし)
- registry.connect.redhat.com (Red Hat Partner Connect プログラムイメージ)

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>
```

ここで、**<registry>[:<port>]/[<namespace>/]<name>:<tag>** はコンテナーイメージの名前です。

たとえば、**registry.redhat.io/ubi9/ubi** コンテナーイメージは以下によって識別されます。

- レジストリーサーバー (**registry.redhat.io**)
- namespace (**ubi9**)
- イメージ名 (**ubi**)

同じイメージにバージョンが複数ある場合は、イメージ名を明示的に指定するタグを追加します。デフォルトでは、Podman は **:latest** タグを使用します (例: **ubi9/ubi:latest**)。

一部のレジストリーでは、**<namespace>** も使用して、異なるユーザーまたは組織によって所有される同じ **<name>** のイメージを区別します。以下に例を示します。

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

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、RHEL サブスクリプション認証情報を使用して認証する必要があります。

3.2. コンテナーレジストリーの設定

registries.conf の設定ファイルで、コンテナーレジストリーの一覧を確認できます。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します。

ユーザーとして、**$HOME/.config/containers/registries.conf** ファイルを作成し、システム全体の設定を上書きします。

```
unqualified-search-registries = ["registry.fedoraproject.org", "registry.access.redhat.com", "docker.io"]
```

デフォルトでは、**podman pull** および **podman search** コマンドは、**unqualified-search-registries** のリストに記載のレジストリーから、その順序でコンテナーイメージを検索します。

ローカルコンテナーレジストリーの設定

TLS 検証なしでローカルコンテナーレジストリーを設定できます。TLS 検証を無効にする方法は 2 つあります。まず、Podman で **--tls-verify=false** オプションを使用できます。次に、**registries.conf** ファイルに **insecure=true** を設定できます。

```
[[registry]]
location="localhost:5000"
insecure=true
```

レジストリー、名前空間、またはイメージのブロック

ローカルシステムがアクセスできないレジストリーを定義できます。**blocked=true** を設定して、特定のレジストリーをブロックできます。

```
[[registry]]
location = "registry.example.org"
blocked = true
```

プレフィックスを **prefix="registry.example.org/namespace"** に設定して、名前空間をブロックすることもできます。たとえば、**podman pull registry.example.org/example/image:latest** コマンドを使用するイメージのプルは、指定したプレフィックスが一致するためブロックされます。

```
[[registry]]
location = "registry.example.org"
prefix="registry.example.org/namespace"
blocked = true
```

注記

prefix はオプションで、デフォルト値は **location** の値と同じです。

prefix="registry.example.org/namespace/image" を設定して、特定のイメージをブロックできます。

```
[[registry]]
location = "registry.example.org"
prefix="registry.example.org/namespace/image"
blocked = true
```

レジストリーのミラーリング

元のレジストリーにアクセスできない場合は、レジストリーミラーを設定できます。たとえば、機密レベルの高い環境で作業するため、インターネットに接続することはできません。指定された順序でアクセスされる複数のミラーを指定できます。たとえば、**podman pull registry.example.com/myimage:latest** コマンドを実行すると、まず **mirror-1.com** が試行され、次に **mirror-2.com** が試行されます。

```
[[registry]]
location="registry.example.com"
[[registry.mirror]]
location="mirror-1.com"
[[registry.mirror]]
location="mirror-2.com"
```

関連情報

- How to manage Linux container registries

3.3. 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。また、Red Hat コンテナーカタログ でイメージを検索することもできます。Red Hat コンテナーレジストリーには、イメージの説明、コンテンツ、ヘルスインデックスなど情報が含まれます。

注記

podman search コマンドは、イメージの存在を判断する信頼できる方法ではありません。v1 および v2 Docker ディストリビューション API の **podman search** 動作は、各レジストリーの実装に固有のものです。一部のレジストリーは、検索をまったくサポートしない場合があります。検索用語を使用しない検索は、v2 API を実装するレジストリーでのみ機能します。**docker search** コマンドにも、同じことが言えます。

このセクションでは、quay.io レジストリーで **postgresql-10** イメージを検索する方法を説明します。

前提条件

- レジストリーが設定されている。

手順

1. レジストリーに対して認証します。

```
# podman login quay.io
```

2. イメージを検索します。
   - 特定のレジストリーで特定のイメージを検索するには、以下を入力します。

```
podman search quay.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
```

- または、特定のレジストリーで提供されるすべてのイメージを表示するには、以下を入力します。

```
# podman search quay.io/
```

- すべてのレジストリー全体でイメージ名を検索するには、以下を入力します。

```
# podman search postgresql-10
```

完全な説明を表示するには、**--no-trunc** オプションをコマンドに渡します。

関連情報

- **podman-search** の man ページ

3.4. レジストリーからのイメージの取得 (プル)

podman pull コマンドを使用して、ローカルシステムにイメージを取得します。

手順

1. registry.redhat.io レジストリーにログインします。

```
$ podman login registry.redhat.io
Username: username
Password: **********
Login Succeeded!
```

2. registry.redhat.io/ubi9/ubi コンテナーイメージをプルします。

```
$ podman pull registry.redhat.io/ubi9/ubi
```

検証手順

- ローカルシステムにプルしたすべてのイメージを一覧表示します。

```
$ podman images
REPOSITORY                    TAG     IMAGE ID      CREATED      SIZE
registry.redhat.io/ubi9/ubi   latest  3269c37eae33  7 weeks ago  208 MB
```

関連情報

- **podman-pull** の man ページ

3.5. 短縮名のエイリアスの設定

Red Hat は、常に完全修飾名でイメージをプルすることを推奨します。ただし、短縮名でイメージをプルすることが一般的です。たとえば、**registry.access.redhat.com/ubi9:latest** の代わりに **ubi9** を使用できます。

registries.conf ファイルにより、短縮名のエイリアスを指定でき、管理者はイメージをプルする場所を完全に制御できます。エイリアスは、**"name" = "value"** の形式で **[aliases]** テーブルで指定されます。エイリアスの一覧は、**/etc/containers/registries.conf.d** ディレクトリーで確認できます。Red Hat は、このディレクトリーでエイリアスのセットを提供します。たとえば、**podman pull ubi9** は、**registry.access.redhat.com/ubi9:latest** である適切なイメージに直接解決します。

以下に例を示します。

```
unqualified-search-registries=["registry.fedoraproject.org", “quay.io"]

[aliases]
"fedora"="registry.fedoraproject.org/fedora"
```

short-names モードは以下のようになります。

- **enforcing**: イメージのプル中に一致するエイリアスが見つからない場合、Podman はユーザーが非修飾レジストリーのいずれかを選択するよう求めます。選択したイメージを正常に取得すると、Podman は、**$HOME/.cache/containers/short-name-aliases.conf** ファイル (ルートレスユーザー) または **/var/cache/containers/short-name-aliases.conf** (root ユーザー) に新しい短縮名のエイリアスを自動的に記録します。ユーザーを要求できない場合 (stdin や stdout など) が TTY ではない場合は、Podman は失敗します。**short-name-aliases.conf** ファイルは、両方が同じエイリアスを指定する場合、**registries.conf** ファイルよりも優先されることに注意してください。

- **permissive**: enforcing モードと似ていますが、ユーザーにプロンプトが表示されないと Podman は失敗しません。代わりに、Podman は指定された順序で修飾されていないすべてのレジストリーを検索します。エイリアスは記録されないことに注意してください。

- **disabled**: すべての非修飾検索レジストリーが指定の順序で試行され、エイリアスは記録されません。

注記

Red Hat では、レジストリー、名前空間、イメージ名、およびタグが含まれる完全修飾イメージ名を使用することを推奨しますします。短縮名を使用する場合は、なりすましの固有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 (つまりレジストリー) を追加します。たとえば、ユーザーは **example.registry.com registry** レジストリーから example コンテナーイメージをプルします。**example.registry.com** が検索リストの最初にない場合には、攻撃者は検索リストの前のほうに別の example イメージを配置することができてしまいます。目的のコンテンツではなく、攻撃者が配置したイメージを間違ってプルして実行する可能性があります。

関連情報

- Container image short names in Podman

3.6. 短縮名のエイリアスを使用したコンテナーイメージのプル

セキュアな短縮名を使用して、ローカルシステムにイメージを取得できます。以下の手順では、**fedora** または **nginx** のコンテナーイメージをプルする方法を説明します。

手順

- コンテナーイメージをプルします。

- **fedora** イメージをプルします。

```
$ podman pull fedora
Resolved "fedora" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull registry.fedoraproject.org/fedora:latest…
...
Storing signatures
...
```

エイリアスが検出され、**registry.fedoraproject.org/fedora** イメージが安全にプルされます。**unqualified-search-registries** の一覧は、**fedora** イメージ名を解決するためには使用されません。

- **nginx** イメージをプルします。

```
$ podman pull nginx
? Please select an image:
registry.access.redhat.com/nginx:latest
registry.redhat.io/nginx:latest
  ▸ docker.io/library/nginx:latest
✔ docker.io/library/nginx:latest
Trying to pull docker.io/library/nginx:latest…
...
Storing signatures
...
```

一致するエイリアスが見つからない場合は、**unqualified-search-registries** 一覧のいずれかを選択するように求められます。選択したイメージが正常にプルされると、新しい短縮名のエイリアスがローカルに記録されます。そうでない場合はエラーが生じます。

検証

- ローカルシステムにプルしたすべてのイメージを一覧表示します。

```
$ podman images
REPOSITORY                           TAG     IMAGE ID      CREATED       SIZE
registry.fedoraproject.org/fedora    latest  28317703decd  12 days ago    184 MB
docker.io/library/nginx              latest  08b152afcfae  13 days ago    137 MB
```

関連情報

- Container image short names in Podman

3.7. イメージの一覧表示

podman images コマンドを使用して、ローカルストレージのイメージを一覧表示します。

前提条件

- プルしたイメージがローカルシステムで利用できる。

手順

- ローカルストレージ内のイメージをすべて一覧表示します。

```
$ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED      SIZE
registry.access.redhat.com/ubi9/ubi  latest  3269c37eae33  6 weeks ago  208 MB
```

関連情報

- **podman-images** の man ページ

3.8. 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プルして実行したら、**podman inspect** コマンドを使用してイメージを調査できます。たとえば、イメージの内容を理解して、イメージ内のソフトウェアを確認します。**podman inspect** コマンドは、名前または ID で識別されるコンテナーおよびイメージに関する情報を表示します。

前提条件

- プルしたイメージがローカルシステムで利用できる。

手順

- **registry.redhat.io/ubi9/ubi** イメージを確認します。

```
$ podman inspect registry.redhat.io/ubi9/ubi
…
 "Cmd": [
        "/bin/bash"
    ],
    "Labels": {
        "architecture": "x86_64",
        "build-date": "2020-12-10T01:59:40.343735",
        "com.redhat.build-host": "cpt-1002.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "ubi9-container",
        "com.redhat.license_terms": "https://www.redhat.com/...,
    "description": "The Universal Base Image is ...
    }
...
```

"Cmd" キーは、コンテナー内で実行するデフォルトのコマンドを指定します。このコマンドは、**podman run** コマンドに引数として指定すると、上書きできます。この ubi9/ubi コンテナーは、**podman run** で起動時に他の引数を指定していない場合には、bash シェルを実行します。**"Entrypoint"** キー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)。

関連情報

- **podman-inspect** の man ページ

3.9. リモートイメージの検証

skopeo inspect コマンドを使用して、システムにイメージをプルする前に、リモートコンテナーレジストリーからイメージに関する情報を表示します。

手順

手順

- **registry.redhat.io/ubi9/ubi-init** イメージを確認します。

```
# skopeo inspect docker://registry.redhat.io/ubi9/ubi-init
{
    "Name": "registry.redhat.io/ubi9/ubi9-init",
    "Digest": "sha256:c6d1e50ab...",
    "RepoTags": [
        ...
        "latest"
    ],
   "Created": "2020-12-10T07:16:37.250312Z",
    "DockerVersion": "1.13.1",
    "Labels": {
        "architecture": "x86_64",
        "build-date": "2020-12-10T07:16:11.378348",
        "com.redhat.build-host": "cpt-1007.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "ubi9-init-container",
        "com.redhat.license_terms": "https://www.redhat.com/en/about/red-hat-end-user-
license-agreements#UBI",
        "description": "The Universal Base Image Init is designed to run an init system as PID 1
for running multi-services inside a container
        ...
```

関連情報

- **skopeo-inspect** の man ページ

3.10. コンテナーイメージのコピー

skopeo copy コマンドを使用して、コンテナーイメージをあるレジストリーから別のレジストリーにコピーできます。たとえば、外部レジストリーのイメージを使用して内部リポジトリーに反映させたり、2 つの異なる場所のイメージレジストリーを同期したりできます。

手順

- **skopeo** コンテナーイメージを **docker://quay.io** から **docker://registry.example.com** にコピーします。

```
$ skopeo copy docker://quay.io/skopeo/stable:latest
docker://registry.example.com/skopeo:latest
```

関連情報

- **skopeo-copy** の man ページ

3.11. ローカルディレクトリーへのイメージレイヤーのコピー

skopeo copy コマンドを使用して、コンテナーイメージのレイヤーをローカルディレクトリーにコピーできます。

手順

1. **/var/lib/images/nginx** ディレクトリーを作成します。

```
$ mkdir -p /var/lib/images/nginx
```

2. **docker://docker.io/nginx:latest** イメージのレイヤーを、新たに作成したディレクトリーにコピーします。

```
$ skopeo copy docker://docker.io/nginx:latest dir:/var/lib/images/nginx
```

検証

- **/var/lib/images/nginx** ディレクトリーの内容を表示します。

```
$ ls /var/lib/images/nginx
08b11a3d692c1a2e15ae840f2c15c18308dcb079aa5320e15d46b62015c0f6f3
...
4fcb23e29ba19bf305d0d4b35412625fea51e82292ec7312f9be724cb6e31ffd  manifest.json
version
```

関連情報

- **skopeo-copy** の man ページ

3.12. タグ付けイメージ

podman tag コマンドを使用して、ローカルイメージに名前を追加します。この追加した名前は、**registryhost/username/NAME:tag** のように複数の部分で構成されます。

前提条件

- プルしたイメージがローカルシステムで利用できる。

手順

1. すべてのイメージを一覧表示します。

```
$ podman images
REPOSITORY                  TAG     IMAGE ID      CREATED      SIZE
registry.redhat.io/ubi9/ubi         latest  3269c37eae33  7 weeks ago  208 MB
```

2. 以下のいずれかを使用して、**myubi** 名を **registry.redhat.io/ubi9/ubi** イメージに割り当てます。

   - イメージ名:

```
$ podman tag registry.redhat.io/ubi9/ubi myubi
```

   - イメージ ID:

```
$ podman tag 3269c37eae33 myubi
```

   どちらのコマンドも、同じ結果になります。

3. すべてのイメージを一覧表示します。

```
$ podman images
REPOSITORY                  TAG     IMAGE ID      CREATED       SIZE
registry.redhat.io/ubi9/ubi         latest  3269c37eae33  2 months ago  208 MB
localhost/myubi             latest  3269c37eae33  2 months ago  208 MB
```

デフォルトのタグがいずれのイメージでも **latest** であることに注意してください。すべてのイメージ名が単一のイメージ ID 3269c37eae33 に割り当てられていることを確認できます。

4. 以下のいずれかを使用して **registry.redhat.io/ubi9/ubi** イメージに **9** タグを追加します。

- イメージ名:

```
$ podman tag registry.redhat.io/ubi9/ubi myubi:9
```

- イメージ ID:

```
$ podman tag 3269c37eae33 myubi:9
```

どちらのコマンドも、同じ結果になります。

5. すべてのイメージを一覧表示します。

```
$ podman images
REPOSITORY                  TAG     IMAGE ID      CREATED       SIZE
registry.redhat.io/ubi9/ubi         latest  3269c37eae33  2 months ago  208 MB
localhost/myubi             latest  3269c37eae33  2 months ago  208 MB
localhost/myubi             9       3269c37eae33  2 months ago  208 MB
```

デフォルトのタグがいずれのイメージでも **latest** であることに注意してください。すべてのイメージ名が単一のイメージ ID 3269c37eae33 に割り当てられていることを確認できます。

registry.redhat.io/ubi9/ubi イメージにタグ付けした後に、コンテナーを実行するオプションが 3 つあります。

- ID (**3269c37eae33**)
- 名前 (**localhost/myubi:latest**)
- 名前 (**localhost/myubi:9**)

関連情報

- **podman-tag** の man ページ

3.13. イメージの保存および読み込み

podman save コマンドを使用して、イメージをコンテナーアーカイブに保存します。別のコンテナー環境に後で復元したり、別のユーザーに送信することもできます。**--format** オプションを使用して、アーカイブ形式を指定できます。サポート対象の形式は以下のとおりです。

- **docker-archive**
- **oci-archive**

- **oci-dir** (oci マニフェストタイプのあるディレクトリー)
- **docker-dir** (v2s2 マニフェストタイプを含むディレクトリー)

デフォルトの形式は **docker-dir** です。

podman load コマンドを使用して、コンテナーイメージアーカイブからコンテナーストレージにイメージを読み込みます。

前提条件

- プルしたイメージがローカルシステムで利用できる。

手順

1. **registry.redhat.io/rhel9/rsyslog** イメージを tarball として保存します。
   - デフォルトの **docker-dir** 形式で以下を行います。

     ```
     $ podman save -o myrsyslog.tar registry.redhat.io/rhel9/rsyslog:latest
     ```

   - **oci-archive** 形式で、**--format** オプションを使用します。

     ```
     $ podman save -o myrsyslog-oci.tar --format=oci-archive registry.redhat.io/rhel9/rsyslog
     ```

     myrsyslog.tar および **myrsyslog-oci.tar** アーカイブは現在のディレクトリーに保存されます。次の手順では、**myrsyslog.tar** tarball で実行されます。
2. **myrsyslog.tar** のファイルタイプを確認します。

   ```
   $ file myrsyslog.tar
   myrsyslog.tar: POSIX tar archive
   ```

3. **myrsyslog.tar** から **registry.redhat.io/rhel9/rsyslog:latest** イメージを読み込むには、以下を実行します。

   ```
   $ podman load -i myrsyslog.tar
   ...
   Loaded image(s): registry.redhat.io/rhel9/rsyslog:latest
   ```

関連情報

- **podman-save** の man ページ

3.14. UBI イメージの再配布

podman push コマンドを使用して、UBI イメージを独自のレジストリーやサードパーティーのレジストリーにプッシュするか、他のと共有します。UBI dnf 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。

前提条件

- プルしたイメージがローカルシステムで利用できる。

手順

1. オプション:**ubi** イメージに名前を追加します。

```
# podman tag registry.redhat.io/ubi9/ubi registry.example.com:5000/ubi9/ubi
```

2. **registry.example.com:5000/ubi9/ubi** イメージをローカルストレージからレジストリーにプッシュします。

```
# podman push registry.example.com:5000/ubi9/ubi
```

重要

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 Container Certification または Red Hat OpenShift Operator Certification の Red Hat Partner Connect Program で認定されていなければ、Red Hat の認定イメージまたは Red Hat のサポートイメージとはみなされません。

3.15. イメージ署名のデフォルト検証

Red Hat Container Registries **/etc/containers/registries.d/registry.access.redhat.com.yaml** および **/etc/containers/registries.d/registry.redhat.io.yaml** ファイルのポリシー YAML ファイルは、**container-tools:latest** モジュールに含まれる **containers-common** パッケージに含まれます。**podman image trust** コマンドを使用して、RHEL でコンテナーイメージの署名を確認します。

手順

1. registry.access.redhat.com に対する既存の信頼範囲を更新します。

```
# podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
registry.access.redhat.com
```

2. オプション。信頼ポリシーの設定を確認するには、**/etc/containers/policy.json** ファイルを表示します。

```
...
"transports": {
  "docker": {
      "registry.access.redhat.com": [
       {
         "type": "signedBy",
         "keyType": "GPGKeys",
         "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
       }
      ]
},
...
```

3. registry.redhat.io に対する既存の信頼範囲を更新します。

```
# podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
registry.redhat.io
```

4. オプション。信頼ポリシーの設定を確認するには、**/etc/containers/policy.json** ファイルを表示します。

```
...
"transports": {
 "docker": {
      "registry.access.redhat.com": [
      {
        "type": "signedBy",
        "keyType": "GPGKeys",
        "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
      }
    ],
     "registry.redhat.io": [
     {
        "type": "signedBy",
        "keyType": "GPGKeys",
        "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
     }
   ]
},
...
```

関連情報

- **podman-image-trust** の man ページ

3.16. イメージの削除

podman rmi コマンドを使用して、ローカルに保存されたコンテナーイメージを削除します。ID または名前を使用して、イメージを削除できます。

手順

1. ローカルシステムにある全イメージの一覧を表示します。

```
$ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED      SIZE
registry.redhat.io/rhel8/rsyslog    latest  4b32d14201de  7 weeks ago  228 MB
registry.redhat.io/ubi8/ubi         latest  3269c37eae33  7 weeks ago  208 MB
localhost/myubi                     X.Y     3269c37eae33  7 weeks ago  208 MB
```

2. すべてのコンテナーを一覧表示します。

```
$ podman ps -a
CONTAINER ID  IMAGE                                 COMMAND         CREATED        STATUS
PORTS   NAMES
7ccd6001166e  registry.redhat.io/rhel8/rsyslog:latest  /bin/rsyslog.sh  6 seconds ago  Up 5
seconds ago          mysyslog
```

registry.redhat.io/rhel8/rsyslog イメージを削除するには、**podman stop** コマンドを使用して、このイメージから実行中のコンテナーをすべて停止する必要があります。コンテナーを ID または名前を使用して停止できます。

3. **mysyslog** コンテナーを停止します。

```
$ podman stop mysyslog
7ccd6001166e9720c47fbeb077e0afd0bb635e74a1b0ede3fd34d09eaf5a52e9
```

4. **registry.redhat.io/rhel8/rsyslog** イメージを削除します。

```
$ podman rmi registry.redhat.io/rhel8/rsyslog
```

- 複数のイメージを削除するには、以下のコマンドを実行します。

```
$ podman rmi registry.redhat.io/rhel8/rsyslog registry.redhat.io/ubi8/ubi
```

- システムからすべてのイメージを削除するには、以下のコマンドを実行します。

```
$ podman rmi -a
```

- 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、**-f** オプションを追加して削除します。

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

関連情報

- **podman-rmi** の man ページ

第4章 コンテナーの使用

コンテナーは、展開されたコンテナーイメージにあるファイルから作成された、実行中プロセスまたは停止プロセスを表します。Podman ツールを使用してコンテナーと連携できます。

4.1. PODMAN RUN コマンド

podman run コマンドは、コンテナーイメージをもとに新しいコンテナーでプロセスを実行します。コンテナーイメージがすでにロードされていない場合は、**podman run** は、そのイメージからコンテナーを起動する前に、**podman pull image** を実行するのと同じ方法で、リポジトリーからイメージおよびイメージの全依存関係を取得します。コンテナープロセスには、独自のファイルシステム、独自のネットワーク、独自のプロセスツリーがあります。

podman run コマンドの形式は次のとおりです。

```
podman run [options] image [command [arg ...]]
```

基本オプションは次のとおりです。

- **--detach (-d)**: コンテナーをバックグラウンドで実行し、新しいコンテナー ID を出力します。
- **--attach (-a)**: フォアグラウンドモードでコンテナーを実行します。
- **--name (-n)**: コンテナーに名前を割り当てます。**--name** で名前がコンテナーに割り当てられていない場合には、ランダムな文字列で名前が生成されます。これはバックグラウンドコンテナーとフォアグラウンドコンテナーの両方で機能します。
- **--rm**:終了時にコンテナーを自動的に削除します。コンテナーが正常に作成または起動できない場合には、削除されない点に注意してください。
- **--tty (-t)**: コンテナーの標準入力に、疑似端末を割り当て、接続します。
- **--interactive (-i)**: 対話式プロセスの場合には、**-i** と **-t** を佩用してコンテナープロセスに端末を割り当てます。**-i -t** は頻繁に **-it** と記述されます。

4.2. ホストからのコンテナーでのコマンド実行

この手順では、**podman run** コマンドを使用して、コンテナーのオペレーティングシステムタイプを表示する方法を説明します。

前提条件

- Podman ツールがインストールされている。

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

手順

1. **cat /etc/os-release** コマンドを使用して、**registry.access.redhat.com/ubi9/ubi** コンテナーイメージをベースとするコンテナーのオペレーティングシステムの種類を表示します。

```
$ podman run --rm registry.access.redhat.com/ubi9/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
...
```

```
ID="rhel"
...
HOME_URL="https://www.redhat.com/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 9"
...
```

2. オプション:すべてのコンテナーを一覧表示します。

```
$ podman ps
CONTAINER ID  IMAGE   COMMAND  CREATED  STATUS  PORTS   NAMES
```

--rm オプションがあるのでコンテナーは表示されません。コンテナーが削除されました。

関連情報

- **podman-run** の man ページ

4.3. コンテナー内でのコマンドの実行

この手順では、**podman run** コマンドを使用して、コンテナーを対話的に実行する方法を説明します。

前提条件

- Podman ツールがインストールされている。

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

手順

1. **registry.redhat.io/ubi9/ubi** イメージに基づいて、**myubi** という名前のコンテナーを実行します。

```
$ podman run --name=myubi -it registry.access.redhat.com/ubi9/ubi /bin/bash
[root@6ccffd0f6421 /]#
```

   - **-i** オプションは対話式のセッションを作成します。**-t** オプションを指定しないと、シェルは開いたままにも拘らず、シェルには何も入力できません。
   - **-t** オプションは、端末セッションを開きます。**-i** オプションを指定しないと、シェルが開き、終了します。

2. システムユーティリティーのセットが含まれる **procps-ng** パッケージをインストールします (例: **ps**、**top**、**uptime** など)。

```
[root@6ccffd0f6421 /]# dnf install procps-ng
```

3. **ps -ef** コマンドを使用して、現在のプロセスを一覧表示します。

```
# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 12:55 pts/0    00:00:00 /bin/bash
```

```
root         31       1  0 13:07 pts/0    00:00:00 ps -ef
```

4. **exit** を実行してコンテナーを終了し、ホストに戻ります。

```
# exit
```

5. オプション:すべてのコンテナーを一覧表示します。

```
$ podman ps
CONTAINER ID  IMAGE                            COMMAND    CREATED         STATUS
PORTS   NAMES
1984555a2c27  registry.redhat.io/ubi9/ubi:latest  /bin/bash  21 minutes ago  Exited (0) 21
minutes ago          myubi
```

コンテナーが終了ステータスであることを確認できます。

関連情報

- **podman-run** の man ページ

4.4. コンテナーの一覧表示

podman ps コマンドを使用して、システムで実行中のコンテナーの一覧を表示します。

前提条件

- Podman ツールがインストールされている。

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

手順

1. **registry.redhat.io/rhel9/rsyslog** イメージをベースとするコンテナーを実行します。

```
$ podman run -d registry.redhat.io/rhel8/rsyslog
```

2. すべてのコンテナーを一覧表示します。
   - 実行中のコンテナーの一覧を表示するには、以下のコマンドを実行します。

```
$ podman ps
CONTAINER ID IMAGE              COMMAND        CREATED       STATUS
PORTS NAMES
74b1da000a11 rhel9/rsyslog /bin/rsyslog.sh 2 minutes ago Up About a minute
musing_brown
```

   - 全コンテナー (実行中または停止) を一覧表示するには以下を実行します。

```
$ podman ps -a
CONTAINER ID IMAGE         COMMAND    CREATED    STATUS              PORTS
NAMES     IS INFRA
d65aecc325a4 ubi9/ubi      /bin/bash  3 secs ago Exited (0) 5 secs ago peaceful_hopper
```

```
false
74b1da000a11 rhel9/rsyslog rsyslog.sh 2 mins ago Up About a minute     musing_brown
false
```

実行され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には、コンテナーがあるので、再起動できます。

関連情報

- **podman-ps** の man ページ

4.5. コンテナーの起動

コンテナーを実行してから停止し、削除しない場合には、ローカルシステムに保存されて再び実行する準備ができます。**podman start** コマンドを使用して、コンテナーを再実行できます。コンテナー ID または名前を使用して、コンテナーを指定できます。

前提条件

- Podman ツールがインストールされている。

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

- 1 つ以上のコンテナーが停止されている。

手順

1. **myubi** コンテナーを起動します。
   - 非対話モードで以下を行います。

     ```
     $ podman start myubi
     ```

     または、**podman start 1984555a2c27** を使用することもできます。

   - 対話モードで、 **-a** (**--attach**) および **-t** (**--interactive**) を使用して、コンテナーの bash シェルと連携します。

     ```
     $ podman start -a -i myubi
     ```

     または、**podman start -a -i 1984555a2c27** を使用することができます。
2. **exit** を実行してコンテナーを終了し、ホストに戻ります。

```
[root@6ccffd0f6421 /]# exit
```

関連情報

- **podman-start** の man ページ

4.6. ホストからのコンテナーの検証

podman inspect コマンドを使用して、既存のコンテナーのメタデータを JSON 形式で検証します。コンテナー ID または名前を使用して、コンテナーを指定できます。

前提条件

- Podman ツールがインストールされている。

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

手順

- ID 64ad95327c74 で定義されるコンテナーを検査します。
  - すべてのメタデータを取得するには、以下のコマンドを実行します。

```
$ podman inspect 64ad95327c74
[
    {
        "Id":
"64ad95327c740ad9de468d551c50b6d906344027a0e645927256cd061049f681",
        "Created": "2021-03-02T11:23:54.591685515+01:00",
        "Path": "/bin/rsyslog.sh",
        "Args": [
            "/bin/rsyslog.sh"
        ],
        "State": {
            "OciVersion": "1.0.2-dev",
            "Status": "running",
            ...
```

  - JSON ファイルから特定の項目 (例: **StartedAt** タイムスタンプ) を取得するには、以下を実行します。

```
$ podman inspect --format='{{.State.StartedAt}}' 64ad95327c74
2021-03-02 11:23:54.945071961 +0100 CET
```

    その情報は階層構造で保存されます。コンテナーの **StartedAt** タイムスタンプ (**StartedAt** は **State** の配下にある) を確認するには、**--format** オプションとコンテナー ID または名前を使用します。

検証する他の項目の例には、以下が含まれます。

- **.path**: コンテナーとともに実行するコマンドを表示します。
- **.Args**: コマンドに指定する引数
- **.Config.ExposedPorts**: コンテナーから公開する TCP または UDP ポート
- **.state.Pid**: コンテナーのプロセス ID を表示します。
- **.HostConfig.PortBindings**: コンテナーからホストへのポートマッピング

関連情報

- **podman-inspect** の man ページ

4.7. LOCALHOST のディレクトリーのコンテナーへのマウント

以下の手順では、コンテナーでホストの **/dev/log** デバイスをマウントして、コンテナー内のログメッセージをホストシステムに公開する方法を説明します。

前提条件

- Podman ツールがインストールされている。

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

手順

1. **log_test** という名前のコンテナーを実行し、コンテナーにホストの **/dev/log** デバイスをマウントします。

```
# podman run --name="log_test" -v /dev/log:/dev/log --rm \
  registry.redhat.io/ubi9/ubi logger "Testing logging to the host"
```

2. **journalctl** ユーティリティーを使用してログを表示します。

```
# journalctl -b | grep Testing
Dec 09 16:55:00 localhost.localdomain root[14634]: Testing logging to the host
```

--rm オプションは、終了時にコンテナーを削除します。

関連情報

- **podman-run** の man ページ

4.8. コンテナーのファイルシステムのマウント

podman mount コマンドを使用して、ホストからアクセス可能な場所に、作業コンテナーの root ファイルシステムをマウントします。

前提条件

- Podman ツールがインストールされている。

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

手順

1. **mysyslog** という名前のコンテナーを実行します。

```
# podman run -d --name=mysyslog registry.redhat.io/rhel9/rsyslog
```

2. オプション:すべてのコンテナーを一覧表示します。

```
# podman ps -a
CONTAINER ID  IMAGE                                COMMAND       CREATED       STATUS
PORTS   NAMES
```

```
c56ef6a256f8  registry.redhat.io/rhel9/rsyslog:latest  /bin/rsyslog.sh  20 minutes ago  Up 20 minutes ago                mysyslog
```

3. **mysyslog** コンテナーをマウントします。

```
# podman mount mysyslog
/var/lib/containers/storage/overlay/990b5c6ddcdeed4bde7b245885ce4544c553d108310e2b797d7be46750894719/merged
```

4. **ls** コマンドを使用して、マウントポイントのコンテンツを表示します。

```
# ls
/var/lib/containers/storage/overlay/990b5c6ddcdeed4bde7b245885ce4544c553d108310e2b797d7be46750894719/merged
bin  boot  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

5. OS バージョンを表示します。

```
# cat
/var/lib/containers/storage/overlay/990b5c6ddcdeed4bde7b245885ce4544c553d108310e2b797d7be46750894719/merged/etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="9 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
...
```

関連情報

- **podman-mount** の man ページ

4.9. 静的 IP を使用したデーモンとしてのサービスの実行

以下の例は、**rsyslog** サービスをバックグラウンドでデーモンプロセスとして実行します。**--ip** オプションは、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 アドレスに設定します (例: 10.88.0.44)。その後、**podman inspect** コマンドを実行して、IP アドレスを適切に設定できます。

前提条件

- Podman ツールがインストールされている。

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

手順

1. コンテナーのネットワークインターフェースを IP アドレス 10.88.0.44 に設定します。

```
# podman run -d --ip=10.88.0.44 registry.access.redhat.com/rhel9/rsyslog
efde5f0a8c723f70dd5cb5dc3d5039df3b962fae65575b08662e0d5b5f9fbe85
```

2. IP アドレスが正しく設定されていることを確認します。

```
# podman inspect efde5f0a8c723 | grep 10.88.0.44
"IPAddress": "10.88.0.44",
```

関連情報

- **podman-inspect** の man ページ
- **podman-run** の man ページ

4.10. 実行中のコンテナー内でのコマンドの実行

podman exec コマンドを使用して、実行中のコンテナーでコマンドを実行し、そのコンテナーを調べます。コンテナーアクティビティーを中断せずに実行中のコンテナーを調査できるので、**podman run** コマンドの代わりに **podman exec** コマンドを使用します。

前提条件

- Podman ツールがインストールされている。

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

- コンテナーが実行されている。

手順

1. **myrsyslog** コンテナー内で **rpm -qa** コマンドを実行し、インストールされているパッケージの一覧を表示します。

```
$ podman exec -it myrsyslog rpm -qa
tzdata-2020d-1.el8.noarch
python3-pip-wheel-9.0.3-18.el8.noarch
redhat-release-8.3-1.0.el8.x86_64
filesystem-3.8-3.el8.x86_64
...
```

2. **myrsyslog** コンテナーで **/bin/bash** コマンドを実行します。

```
$ podman exec -it myrsyslog /bin/bash
```

3. システムユーティリティーのセットが含まれる **procps-ng** パッケージをインストールします (例: **ps**、**top**、**uptime** など)。

```
# dnf install procps-ng
```

4. コンテナーを検査します。
   - システムにある全プロセスを一覧表示するには、以下のコマンドを実行します。

```
# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 10:23 ?        00:00:01 /usr/sbin/rsyslogd -n
root           8       0  0 11:07 pts/0    00:00:00 /bin/bash
root          47       8  0 11:13 pts/0    00:00:00 ps -ef
```

- ファイルシステムのディスク領域の使用量を表示するには、次のコマンドを実行します。

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
fuse-overlayfs   27G  7.1G   20G  27% /
tmpfs            64M     0   64M   0% /dev
tmpfs           269M  936K  268M   1% /etc/hosts
shm              63M     0   63M   0% /dev/shm
...
```

- システム情報を表示するには、以下のコマンドを実行します。

```
# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

- MB 単位でメモリの空き容量を表示するには、次のコマンドを実行します。

```
# free --mega
total        used        free      shared  buff/cache   available
Mem:      2818         615        1183          12        1020        1957
Swap:     3124           0        3124
```

関連情報

- **podman-exec** の man ページ

4.11. 2 つのコンテナー間でのファイルの共有

コンテナーが削除されても、ボリュームを使用してコンテナー内のデータを永続化できます。ボリュームは、複数のコンテナー間でのデータ共有に使用できます。ボリュームとは、ホストマシンに保存されているフォルダーです。ボリュームはコンテナーとホスト間で共有できます。

主な利点は以下のとおりです。

- ボリュームはコンテナー間で共有できます。
- ボリュームは、他と比べるとバックアップまたは移行が簡単です。
- ボリュームを使用するとコンテナーのサイズが増えません。

前提条件

- Podman ツールがインストールされている。

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

手順

1. ボリュームを作成します。

```
$ podman volume create hostvolume
```

2. ボリュームに関する情報を表示します。

```
$ podman volume inspect hostvolume
[
    {
        "name": "hostvolume",
        "labels": {},
        "mountpoint":
"/home/username/.local/share/containers/storage/volumes/hostvolume/_data",
        "driver": "local",
        "options": {},
        "scope": "local"
    }
]
```

volumes ディレクトリーにボリュームが作成されることに注意してください。**$ mntPoint=$(podman volume inspect hostvolume --format {{.Mountpoint}})** で、変数へのマウントポイントパスを保存して操作を簡素化できます。

sudo podman volume create hostvolume を実行すると、マウントポイントが **/var/lib/containers/storage/volumes/hostvolume/_data** に変わります。

3. **mntPoint** 変数に保管されたパスを使用して、ディレクトリー内にテキストファイルを作成します。

```
$ echo "Hello from host" >> $mntPoint/host.txt
```

4. **mntPoint** 変数で定義されたディレクトリー内の全ファイルを一覧表示します。

```
$ ls $mntPoint/
host.txt
```

5. **myubi1** という名前のコンテナーを実行し、ホストのボリューム名 **hostvolume** で定義したディレクトリーをコンテナーの **/containervolume1** ディレクトリーにマッピングします。

```
$ podman run -it --name myubi1 -v hostvolume:/containervolume1
registry.access.redhat.com/ubi9/ubi /bin/bash
```

mntPoint 変数 (**-v $mntPoint:/containervolume1**) で定義したボリュームパスを使用する場合には、**podman volume prune** コマンドを実行すると未使用のボリュームが削除され、データが失われる場合がある点に注意してください。常に **-v hostvolume_name:/containervolume_name** を使用します。

6. コンテナー上にある共有ボリューム内のファイルを一覧表示します。

```
# ls /containervolume1
host.txt
```

ホスト上で作成した **host.txt** ファイルが表示されます。

7. **/containervolume1** ディレクトリーにテキストファイルを作成します。

```
# echo "Hello from container 1" >> /containervolume1/container1.txt
```

8. **CTRL+p** および **CTRL+q** を使用してコンテナーからデタッチします。

9. ホスト上にある共有ボリューム内のファイルを一覧表示します。以下の 2 つのファイルが表示されるはずです。

```
$ ls $mntPoint
container1.rxt  host.txt
```

この時点で、コンテナーとホスト間でファイルを共有しています。2 つのコンテナー間でファイルを共有するには、**myubi2** という名前の別のコンテナーを実行します。

10. **myubi2** という名前のコンテナーを実行し、ホストのボリューム名 **hostvolume** で定義したディレクトリーをコンテナーの **/containervolume2** ディレクトリーにマッピングします。

```
$ podman run -it --name myubi2 -v hostvolume:/containervolume2
registry.access.redhat.com/ubi9/ubi  /bin/bash
```

11. コンテナー上にある共有ボリューム内のファイルを一覧表示します。

```
# ls /containervolume2
container1.txt host.txt
```

ホストで作成した **host.txt** ファイルと、**myubi1** コンテナー内に作成した **container1.txt** ファイルが表示されます。

12. **/containervolume2** ディレクトリーにテキストファイルを作成します。

```
# echo "Hello from container 2" >> /containervolume2/container2.txt
```

13. **CTRL+p** および **CTRL+q** を使用してコンテナーからデタッチします。

14. ホスト上にある共有ボリューム内のファイルを一覧表示します。以下の 3 つのファイルが表示されるはずです。

```
$ ls $mntPoint
container1.rxt  container2.txt host.txt
```

関連情報

- **podman-volume** の man ページ

4.12. コンテナーのエクスポートおよびインポート

podman export コマンドを使用して、実行中のコンテナーのファイルシステムをローカルマシンの tarball にエクスポートできます。たとえば、頻繁に使用しない大規模なコンテナーがある場合、スナップショットを保存して後で復元できるようにする場合には、**podman export** コマンドを使用して、実行中のコンテナーの現在のスナップショットを tarball にエクスポートできます。

podman import コマンドを使用して tarball をインポートし、ファイルシステムイメージとして保存できます。これにより、このファイルシステムイメージを実行するか、他のイメージのレイヤーとして使用できます。

前提条件

- Podman ツールがインストールされている。

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

手順

1. **registry.access.redhat.com/ubi9/ubi** イメージに基づいて、**myubi** コンテナーを実行します。

```
$ podman run -dt --name=myubi registry.access.redhat.com/9/ubi
```

2. オプション:すべてのコンテナーを一覧表示します。

```
$ podman ps -a
CONTAINER ID  IMAGE                                COMMAND         CREATED     STATUS
PORTS   NAMES
a6a6d4896142  registry.access.redhat.com/9:latest   /bin/bash       7 seconds ago  Up 7
seconds ago          myubi
```

3. **myubi** コンテナーに割り当てます。

```
$ podman attach myubi
```

4. **testfile** という名前のファイルを作成します。

```
[root@a6a6d4896142 /]# echo "hello" > testfile
```

5. **CTRL+p** および **CTRL+q** を使用してコンテナーからデタッチします。

6. ローカルマシンで、**myubi** のファイルシステムを **myubi-container.tar** としてエクスポートします。

```
$ podman export -o myubi.tar a6a6d4896142
```

7. オプション:現在のディレクトリーの内容を一覧表示します。

```
$ ls -l
-rw-r--r--. 1 user user 210885120 Apr  6 10:50 myubi-container.tar
...
```

8. オプション:**myubi-container** ディレクトリーを作成し、**myubi-container.tar** アーカイブからすべてのファイルを展開します。ツリー形式で **myubi-directory** の内容を一覧表示します。

```
$ mkdir myubi-container
$ tar -xf myubi-container.tar -C myubi-container
$ tree -L 1 myubi-container
├── bin -> usr/bin
├── boot
├── dev
├── etc
├── home
├── lib -> usr/lib
├── lib64 -> usr/lib64
├── lost+found
├── media
├── mnt
```

```
├── opt
├── proc
├── root
├── run
├── sbin -> usr/sbin
├── srv
├── sys
├── testfile
├── tmp
├── usr
└── var

20 directories, 1 file
```

myubi-container.tar にコンテナーのファイルシステムが含まれていることを確認できます。

9. **myubi.tar** をインポートして、ファイルシステムイメージとして保存します。

```
$ podman import myubi.tar myubi-imported
Getting image source signatures
Copying blob 277cab30fe96 done
Copying config c296689a17 done
Writing manifest to image destination
Storing signatures
c296689a17da2f33bf9d16071911636d7ce4d63f329741db679c3f41537e7cbf
```

10. すべてのイメージを一覧表示します。

```
$ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED         SIZE
docker.io/library/myubi-imported    latest  c296689a17da  51 seconds ago  211 MB
```

11. **testfile** ファイルの内容を表示します。

```
$ podman run -it  --name=myubi-imported docker.io/library/myubi-imported cat testfile
hello
```

関連情報

- **podman-export** man ページ
- **podman-import** man ページ

4.13. コンテナーの停止

実行中のコンテナーを停止するには、**podman stop** コマンドを使用します。コンテナー ID または名前を使用して、コンテナーを指定できます。

前提条件

- Podman ツールがインストールされている。

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

- 1 つ以上のコンテナーが実行中である。

手順

- **myubi** コンテナーを停止します。
  - コンテナー名を使用する場合

```
$ podman stop myubi
```

  - コンテナー ID を使用する場合

```
$ podman stop 1984555a2c27
```

端末セッションに接続されている、実行中のコンテナーを停止するには、コンテナーで **exit** コマンドを入力してください。

podman stop コマンドは、SIGTERM シグナルを送信し、実行中のコンテナーを終了します。定義された期間 (デフォルトでは 10 秒) 後にコンテナーが停止しない場合は、Podman は SIGKILL シグナルを送信します。

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、別のシグナルをコンテナーに送信することもできます。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
```

関連情報

- **podman-stop** の man ページ
- **podman-kill** の man ページ

4.14. コンテナーの削除

podman rm コマンドを使用して、コンテナーを削除します。コンテナー ID または名前でコンテナーを指定できます。

前提条件

- Podman ツールがインストールされている。

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

- 1 つ以上のコンテナーが停止されている。

手順

1. 全コンテナー (実行中または停止) を一覧表示するには以下を実行します。

```
$ podman ps -a
```

```
CONTAINER ID IMAGE          COMMAND    CREATED    STATUS                  PORTS NAMES
IS INFRA
d65aecc325a4 ubi9/ubi       /bin/bash  3 secs ago Exited (0) 5 secs ago peaceful_hopper false
74b1da000a11 rhel9/rsyslog rsyslog.sh 2 mins ago Up About a minute      musing_brown
false
```

2. コンテナーを削除します。

- **peaceful_hopper** を削除するには以下を実行します。

```
$ podman rm peaceful_hopper
```

peaceful_hopper コンテナーが終了ステータスであったことに注意してください。コンテナーが停止されているので、即座に削除できます。

- **musing_brown** コンテナーを削除するには、まずコンテナーを停止してから削除します。

```
$ podman stop musing_brown
$ podman rm musing_brown
```

注記

- 複数のコンテナーを削除するには、以下を実行します。

```
$ podman rm clever_yonath furious_shockley
```

- ローカルシステムからすべてのコンテナーを削除するには、以下のコマンドを実行します。

```
$ podman rm -a
```

関連情報

- **podman-rm** の man ページ

4.15. RUNC コンテナーランタイム

runcコンテナーランタイムは、Open Container Initiative (OCI) コンテナーランタイム仕様の軽量で移植可能な実装です。runc ランタイムは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。

4.16. CRUN コンテナーランタイム

crun は、C 言語で書かれた高速および低メモリーフットプリントの OCI コンテナーランタイムで、crun バイナリーはrunc バイナリーの最大 1/50 のサイズで、2 倍の速度です。crun を使用して、コンテナーの実行時に最小限のプロセス数を設定することもできます。crun ランタイムは OCI フックもサポートしています。

crun の追加機能には、以下が含まれます。

- ルートレスコンテナーのグループによるファイルの共有
- OCI フックの標準出力および標準エラーの制御
- cgroup v2 での古いバージョンの systemd の実行
- 他のプログラムによって使用される C ライブラリー
- 拡張性
- 移植性

関連情報

- An introduction to crun, a fast and low-memory footprint container runtime

4.17. RUNC および CRUN でのコンテナーの実行

runc または crun では、コンテナーはバンドルを使用して設定されます。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。

注記

<runtime> は crun または runc です。

手順

1. **registry.access.redhat.com/ubi9/ubi** コンテナーイメージをプルします。

```
# podman pull registry.access.redhat.com/ubi9/ubi
```

2. **registry.access.redhat.com/ubi9/ubi** イメージを **rhel.tar** アーカイブにエクスポートします。

```
# podman export $(podman create registry.access.redhat.com/ubi9/ubi) > rhel.tar
```

3. **bundle/rootfs** ディレクトリーを作成します。

```
# mkdir -p bundle/rootfs
```

4. **rhel.tar** アーカイブを **bundle/rootfs** ディレクトリーに展開します。

```
# tar -C bundle/rootfs -xf rhel.tar
```

5. バンドル用に **config.json** という名前の新規仕様ファイルを作成します。

```
# <runtime> spec -b bundle
```

- **-b** オプションは、バンドルのディレクトリーを指定します。デフォルト値は現在のディレクトリーです。

6. オプション。設定を変更します。

```
# vi bundle/config.json
```

7. バンドル用に **myubi** という名前のコンテナーのインスタンスを作成します。

```
# <runtime>  create -b bundle/ myubi
```

8. **myubi** コンテナーを起動します。

```
# <runtime>  start myubi
```

注記

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。コンテナーの新規インスタンスを起動するには、**# <runtime> start <container_name>**を実行します。

検証

- **<runtime>** によって起動したコンテナーを一覧表示します。

```
# <runtime> list
ID            PID         STATUS      BUNDLE          CREATED                        OWNER
myubi         0           stopped     /root/bundle    2021-09-14T09:52:26.659714605Z   root
```

関連情報

- **crun** の man ページ
- **runc** の man ページ
- An introduction to crun, a fast and low-memory footprint container runtime

4.18. コンテナーランタイムの一時的な変更

podman run コマンドに **--runtime** オプションを指定して使用して、コンテナーのランタイムを変更できます。

注記

<runtime> は crun または runc です。

前提条件

- Podman ツールがインストールされている。

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

手順

- **registry.access.redhat.com/ubi9/ubi** コンテナーイメージをプルします。

```
$ podman pull registry.access.redhat.com/ubi9/ubi
```

1. **--runtime** オプションを使用してコンテナーランタイムを変更します。

```
$ podman run --name=myubi -dt --runtime=<runtime> ubi9
bashe4654eb4df12ac031f1d0f2657dc4ae6ff8eb0085bf114623b66cc664072e69b
```

2. オプション。すべてのイメージを一覧表示します。

```
$ podman ps -a
CONTAINER ID  IMAGE                                 COMMAND  CREATED       STATUS
PORTS   NAMES
e4654eb4df12  registry.access.redhat.com/ubi9:latest  bash     4 seconds ago  Up 4
seconds ago          myubi
```

検証

- OCI ランタイムが myubi コンテナーで **<runtime>** に設定されていることを確認します。

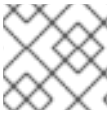
```
$ podman inspect myubi --format "{{.OCIRuntime}}"
<runtime>
```

関連情報

- An introduction to crun, a fast and low-memory footprint container runtime

4.19. コンテナーランタイムの永続的な変更

コンテナーランタイムおよびそのオプションを、**/etc/containers/containers.conf** 設定ファイル(root ユーザーとして)または**$HOME/.config/containers/containers.conf**設定ファイル(非rootユーザーとして)で設定できます。



注記

<runtime> は crun または runc ランタイムです。

前提条件

- Podman ツールがインストールされている。

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

手順

- **/etc/containers/containers.conf** ファイルでランタイムを変更します。

```
# vim /etc/containers/containers.conf
[engine]
runtime = "<runtime>"
```

- myubi という名前のコンテナーを実行します。

```
# podman run --name=myubi -dt ubi9 bash
Resolved "ubi9" as an alias (/etc/containers/registries.conf.d/001-rhel-shortnames.conf)
Trying to pull registry.access.redhat.com/ubi9:latest…
...
Storing signatures
```

検証

- OCI ランタイムが **myubi** コンテナーで **<runtime>** に設定されていることを確認します。

```
# podman inspect myubi --format "{{.OCIRuntime}}"
<runtime>
```

関連情報

- An introduction to crun, a fast and low-memory footprint container runtime
- **containers.conf** の man ページ

4.20. 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。

第5章 POD の使用

コンテナーは、Podman、Skopeo、および Buildah のコンテナーツールで管理できる最小単位です。Podman Pod は、1 つ以上のコンテナーのグループです。Pod の概念は Kubernetes により導入されました。Podman Pod は Kubernetes 定義に似ています。Pod は、OpenShift または Kubernetes 環境で作成、デプロイ、管理できる最小のコンピュート単位です。すべての Podman Pod には infra コンテナーが含まれます。このコンテナーは Pod に関連付けられた namespace を保持し、Podman が他のコンテナーを Pod に接続できるようにします。これにより、Pod を稼働したまま、Pod 内のコンテナーの起動や停止が可能になります。デフォルトの infra コンテナーは、**registry.access.redhat.com/ubi9/pause** イメージ上にあります。

5.1. POD の作成

以下の手順では、コンテナーが 1 つ含まれる Pod を作成する方法を説明します。

前提条件

- Podman ツールがインストールされている。

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

手順

1. 空の Pod を作成します。

```
$ podman pod create --name mypod
223df6b390b4ea87a090a4b5207f7b9b003187a6960bd37631ae9bc12c433aff
The pod is in the initial state Created.
```

   Pod の初期状態は Created です。

2. オプション:すべての Pod を一覧表示します。

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED               # OF CONTAINERS   INFRA ID
223df6b390b4   mypod   Created   Less than a second ago   1              3afdcd93de3e
```

   Pod にはコンテナーが 1 つあることに注意してください。

3. オプション:関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                COMMAND  CREATED              STATUS   PORTS
NAMES              POD
3afdcd93de3e  registry.access.redhat.com/ubi9/pause          Less than a second ago
Created        223df6b390b4-infra  223df6b390b4
```

   podman ps コマンドからの Pod ID と **podman pod ps** コマンドの Pod ID が一致することが確認できます。デフォルトの infra コンテナーは **registry.access.redhat.com/ubi9/pause** イメージをベースとしています。

4. **mypod** という名前の既存 Pod 内で、名前が **myubi** のコンテナーを実行します。

```
$ podman run -dt --name myubi --pod mypod registry.access.redhat.com/ubi9/ubi  /bin/bash
5df5c48fea87860cf75822ceab8370548b04c78be9fc156570949013863ccf71
```

5. オプション:すべての Pod を一覧表示します。

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED               # OF CONTAINERS   INFRA ID
223df6b390b4  mypod   Running   Less than a second ago  2               3afdcd93de3e
```

Pod にはコンテナーが 2 つあることが分かります。

6. オプション:関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED
STATUS                   PORTS  NAMES             POD
5df5c48fea87  registry.access.redhat.com/ubi9/ubi:latest  /bin/bash  Less than a second ago
Up Less than a second ago         myubi             223df6b390b4
3afdcd93de3e  registry.access.redhat.com/ubi9/pause                    Less than a
second ago  Up Less than a second ago         223df6b390b4-infra  223df6b390b4
```

関連情報

- **podman-pod-create** man ページ
- Podman: Managing pods and containers in a local container runtime

5.2. POD 情報の表示

この手順では、Pod 情報を表示する方法を説明します。

前提条件

- Podman ツールがインストールされている。

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

- Pod が作成されている。詳細は、「Pod の作成」を参照してください。

手順

- Pod で実行されているアクティブなプロセスを表示します。
  - Pod 内のコンテナーで実行中のプロセスを表示するには、以下を入力します。

```
$ podman pod top mypod
USER   PID   PPID   %CPU    ELAPSED         TTY     TIME   COMMAND
0      1     0      0.000   24.077433518s   ?       0s     /pause
root   1     0      0.000   24.078146025s   pts/0   0s     /bin/bash
```

  - 1 つ以上の Pod のコンテナーにおけるリソース使用状況の統計をライブストリームで表示するには、以下を入力します。

```
$ podman pod stats -a --no-stream
ID            NAME             CPU %   MEM USAGE / LIMIT   MEM %   NET IO    BLOCK IO
PIDS
a9f807ffaacd  frosty_hodgkin   --      3.092MB / 16.7GB    0.02%   -- / --   -- / --    2
3b33001239ee  sleepy_stallman  --      -- / --             --      -- / --   -- / --    --
```

- Pod に関する情報を表示するには、以下を入力します。

```
$ podman pod inspect mypod
{
   "Id": "db99446fa9c6d10b973d1ce55a42a6850357e0cd447d9bac5627bb2516b5b19a",
   "Name": "mypod",
   "Created": "2020-09-08T10:35:07.536541534+02:00",
   "CreateCommand": [
      "podman",
      "pod",
      "create",
      "--name",
      "mypod"
   ],
   "State": "Running",
   "Hostname": "mypod",
   "CreateCgroup": false,
   "CgroupParent": "/libpod_parent",
   "CgroupPath":
"/libpod_parent/db99446fa9c6d10b973d1ce55a42a6850357e0cd447d9bac5627bb2516b5
b19a",
   "CreateInfra": false,
   "InfraContainerID":
"891c54f70783dcad596d888040700d93f3ead01921894bc19c10b0a03c738ff7",
   "SharedNamespaces": [
      "uts",
      "ipc",
      "net"
   ],
   "NumContainers": 2,
   "Containers": [
      {
         "Id":
"891c54f70783dcad596d888040700d93f3ead01921894bc19c10b0a03c738ff7",
         "Name": "db99446fa9c6-infra",
         "State": "running"
      },
      {
         "Id":
"effc5bbcfe505b522e3bf8fbb5705a39f94a455a66fd81e542bcc27d39727d2d",
         "Name": "myubi",
         "State": "running"
      }
   ]
}
```

Pod 内のコンテナーについての情報を確認できます。

関連情報

- **podman pod top** man ページ
- **podman-pod-stats** man ページ
- **podman-pod-inspect** man ページ

5.3. POD の停止

1 つ以上の Pod を停止するには、**podman pod stop** コマンドを使用します。

前提条件

- Podman ツールがインストールされている。

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

- Pod が作成されている。詳細は、「Pod の作成」を参照してください。

手順

1. Pod **mypod** を停止します。

```
$ podman pod stop mypod
```

2. オプション:関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED            STATUS
PORTS   NAMES              POD ID        PODNAME
5df5c48fea87  registry.redhat.io/ubi9/ubi:latest  /bin/bash  About a minute ago  Exited (0) 7
seconds ago          myubi              223df6b390b4  mypod

3afdcd93de3e  registry.access.redhat.com/9/pause                     About a minute ago
Exited (0) 7 seconds ago          8a4e6527ac9d-infra  223df6b390b4  mypod
```

Pod **mypod** およびコンテナー **myubi** のステータスが「Exited」であることが分かります。

関連情報

- **podman-pod-stop** man ページ

5.4. POD の削除

停止されている Pod またはコンテナーを削除するには、**podman pod rm** コマンドを使用します。

前提条件

- Podman ツールがインストールされている。

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

- Pod が作成されている。詳細は、「Pod の作成」を参照してください。

- Pod が停止されている。詳細は、「Pod の停止」を参照してください。

手順

1. 以下を入力して、Pod **mypod** を削除します。

```
$ podman pod rm mypod
223df6b390b4ea87a090a4b5207f7b9b003187a6960bd37631ae9bc12c433aff
```

   Pod を削除すると、その Pod 中の全コンテナーが自動的に削除されることに注意してください。

2. オプション:すべてのコンテナーおよび Pod が削除されたことを確認します。

```
$ podman ps
$ podman pod ps
```

関連情報

- **podman-pod-rm** man ページ

第6章 コンテナーネットワークの管理

この章では、コンテナーネットワークの管理方法について説明します。

6.1. コンテナーネットワークのリストアップ

Podman には、ルートレスルートフルという 2 つのネットワークの動作があります。

- ルートレスネットワーク: ネットワークは自動的に設定され、コンテナーには IP アドレスがありません。
- ルートフルネットワーク: コンテナーには IP アドレスがあります。

手順

- ルートユーザーで全ネットワークを一覧表示します。

```
# podman network ls
NETWORK ID    NAME        VERSION     PLUGINS
2f259bab93aa  podman      0.4.0       bridge,portmap,firewall,tuning
```

  - デフォルトでは、Podman はブリッジネットワークを提供します。
  - ルートレスユーザーのネットワーク一覧は、ルートフルユーザーと同じです。

関連情報

- **podman-network-ls** の man ページ

6.2. ネットワークの検査

podman network ls コマンドでリストアップされた特定のネットワークの IP 範囲、有効なプラグイン、ネットワークのタイプなどを表示します。

手順

- デフォルトの **Podman** ネットワークを検査します。

```
$ podman network inspect podman
[
   {
      "cniVersion": "0.4.0",
      "name": "podman",
      "plugins": [
         {
            "bridge": "cni-podman0",
            "hairpinMode": true,
            "ipMasq": true,
            "ipam": {
               "ranges": [
                  [
                     {
                        "gateway": "10.88.0.1",
                        "subnet": "10.88.0.0/16"
```

```
                    }
                ]
            ],
            "routes": [
                {
                    "dst": "0.0.0.0/0"
                }
            ],
            "type": "host-local"
        },
        "isGateway": true,
        "type": "bridge"
    },
    {
        "capabilities": {
            "portMappings": true
        },
        "type": "portmap"
    },
    {
        "type": "firewall"
    },
    {
        "type": "tuning"
    }
  ]
 }
]
```

IP 範囲、有効なプラグイン、ネットワークの種類など、ネットワークの設定を確認できます。

関連情報

- **podman-network-inspect** の man ページ

6.3. ネットワークの作成

新しいネットワークを作成するには、**Podman network create** コマンドを使用します。

注記

デフォルトでは、Podman は外部ネットワークを作成します。**podman network create --internal** コマンドで内部ネットワークを作成することができます。内部ネットワーク内のコンテナーは、ホスト上の他のコンテナーと通信できますが、ホスト外のネットワークに接続したり、ホストから到達したりすることはできません。

手順

- **mynet** という名前の外部ネットワークを作成します。

```
# podman network create mynet
/etc/cni/net.d/mynet.conflist
```

検証

- すべてのネットワークをリストアップします。

```
# podman network ls
NETWORK ID    NAME        VERSION     PLUGINS
2f259bab93aa  podman      0.4.0       bridge,portmap,firewall,tuning
11c844f95e28  mynet       0.4.0       bridge,portmap,firewall,tuning,dnsname
```

作成された **mynet** ネットワークとデフォルトの **Podman** ネットワークが表示されます。

注記

podman network create コマンドを使用して新しい外部ネットワークを作成すると、DNS プラグインがデフォルトで有効になります。

関連情報

- **podman-network-create** man ページ

6.4. コンテナーのネットワークへの接続

コンテナーをネットワークに接続するには、**Podman network connect** コマンドを使用します。

前提条件

- **podman network create** コマンドでネットワークが作成さている。
- コンテナーが作成されている。

手順

- **mycontainer** という名前のコンテナーを **mynet** という名前のネットワークに接続します。

```
# podman network connect mynet mycontainer
```

検証

- **mycontainer** が **mynet** ネットワークに接続されていることを確認します。

```
# podman inspect --format='{{.NetworkSettings.Networks}}' mycontainer
map[podman:0xc00042ab40 mynet:0xc00042ac60]
```

mycontainer が **mynet** と **podman** のネットワークに接続されていることがわかります。

関連情報

- **podman-network-connect** の man ページ

6.5. コンテナーのネットワークからの切断

コンテナーをネットワークから切断するには、**podman network disconnect** コマンドを使用します。

前提条件

- **podman network create** コマンドでネットワークが作成さている。
- コンテナーがネットワークに接続されている。

手順

- **mycontainer** というコンテナーを **mynet** というネットワークから切り離します。

```
# podman network disconnect mynet mycontainer
```

検証

- **mycontainer** が **mynet** ネットワークから切断されていることを確認します。

```
# podman inspect --format='{{.NetworkSettings.Networks}}' mycontainer
map[podman:0xc000537440]
```

mycontainer が **mynet** ネットワークから切断され、**mycontainer** がデフォルトの **Podman** ネットワークにのみ接続されていることがわかります。

関連情報

- **podman-network-disconnect** の man ページ

6.6. ネットワークの削除

指定したネットワークを削除するには、**podman network rm** コマンドを使用します。

手順

1. すべてのネットワークをリストアップします。

```
# podman network ls
NETWORK ID    NAME        VERSION     PLUGINS
2f259bab93aa  podman      0.4.0       bridge,portmap,firewall,tuning
11c844f95e28  mynet       0.4.0       bridge,portmap,firewall,tuning,dnsname
```

2. **mynet** ネットワークを削除します。

```
# podman network rm mynet
mynet
```

注記

削除されたネットワークに関連するコンテナーがある場合は、**Podman network rm -f** コマンドを使用してコンテナーと Pod を削除する必要があります。

検証

- **mynet** ネットワークが削除されたかどうかを確認します。

```
# podman network ls
NETWORK ID    NAME        VERSION     PLUGINS
2f259bab93aa  podman      0.4.0       bridge,portmap,firewall,tuning
```

関連情報

- **podman-network-rm** の man ページ

6.7. 未使用の全ネットワークの削除

podman network prune で、使われていないネットワークをすべて削除してください。未使用のネットワークとは、コンテナーが接続されていないネットワークのことです。**podman network prune** コマンドでは、デフォルトの **podman** ネットワークは削除されません。

手順

- 未使用のネットワークをすべて削除します。

```
# podman network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
```

検証

- すべてのネットワークが削除されたことを確認します。

```
# podman network ls
NETWORK ID    NAME        VERSION     PLUGINS
2f259bab93aa  podman      0.4.0       bridge,portmap,firewall,tuning
```

関連情報

- **podman-network-prune** の man ページ

第7章 コンテナー間の通信

この章では、コンテナー間の通信方法について説明します。

7.1. ネットワークモードとレイヤー

Podman には、いくつかの異なるネットワークモードがあります。

- **bridge** - デフォルトのブリッジネットワーク上に別のネットワークを作成します。
- **container:<id>** - id が **<id>の** コンテナーと同じネットワークを使用します。
- **host** - ホストネットワークスタックを使用します。
- **network-id** - **podman** network create コマンドで作成されたユーザー定義のネットワークを使用します。
- **private** - コンテナーの新しいネットワークを作成します。
- **slirp4nets** - ルートレスコンテナーのデフォルトオプションである slirp4netns を使ってユーザーネットワークスタックを作成します。

注記

ホストモードでは、D-bus (コンテナーはプロセス間通信 (IPC) システム) などのローカルシステムサービスにフルアクセスできるため、安全でないと考えられています。

7.2. コンテナーのネットワーク設定の検査

podman inspect コマンドに **--format** オプションを付けると、**podman inspect** の出力から個々の項目を表示できます。

手順

1. コンテナーの IP アドレスを表示します。

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' containerName
```

2. コンテナーが接続されている全ネットワークを表示します。

```
# podman inspect --format='{{.NetworkSettings.Networks}}' containerName
```

3. ポートマッピングを表示します。

```
# podman inspect --format='{{.NetworkSettings.Ports}}' containerName
```

関連情報

- **podman-inspect** の man ページ

7.3. コンテナーとアプリケーション間の通信

コンテナーとアプリケーションの間で通信を行うことができます。アプリケーションのポートは、リスニング状態かオープン状態のどちらかです。これらのポートは自動的にコンテナーネットワークに公開されるため、このようなネットワークを使用してコンテナーに到達できます。デフォルトでは、Web サーバーはポート 80 でリッスンします。この手順で、**myubi** コンテナーは **web-container** アプリケーションと通信を行います。

手順

1. **web-container** という名前のコンテナーを起動します。

```
# podman run -dt --name=web-container docker.io/library/httpd
```

2. すべてのコンテナーを一覧表示します。

```
# podman ps -a

CONTAINER ID  IMAGE                           COMMAND         CREATED        STATUS
PORTS       NAMES
b8c057333513  docker.io/library/httpd:latest  httpd-foreground  4 seconds ago  Up 5 seconds
ago             web-container
```

3. コンテナーを点検し、IP アドレスを表示します。

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' web-container

10.88.0.2
```

4. **myubi** コンテナーを実行し、Web サーバーが動作していることを確認します。

```
# podman run -it --name=myubi ubi9/ubi curl 10.88.0.2:80

<html><body><h1>It works!</h1></body></html>
```

7.4. コンテナーとホスト間の通信

デフォルトでは、**Podman** ネットワークはブリッジネットワークです。つまり、ネットワークデバイスがコンテナーネットワークとホストネットワークをつなぎます。

前提条件

- **web-container** が動作している。詳細は、コンテナーとアプリケーションの間の通信 のセクションを参照してください。

手順

1. ブリッジが設定されていることを確認します。

```
# podman network inspect podman | grep bridge

    "bridge": "cni-podman0",
    "type": "bridge"
```

2. ホストネットワークの設定を表示します。

```
# ip addr show cni-podman0

6: cni-podman0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether 62:af:a1:0a:ca:2e brd ff:ff:ff:ff:ff:ff
    inet 10.88.0.1/16 brd 10.88.255.255 scope global cni-podman0
       valid_lft forever preferred_lft forever
    inet6 fe80::60af:a1ff:fe0a:ca2e/64 scope link
       valid_lft forever preferred_lft forever
```

web-container には **cni-podman0** ネットワークの IP が指定されており、そのネットワークがホストにブリッジされていることがわかります。

3. **web-containe** をチェックし、その IP アドレスを表示します。

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' web-container

10.88.0.2
```

4. ホストから直接 **web-container** にアクセスします。

```
$ curl 10.88.0.2:80

<html><body><h1>It works!</h1></body></html>
```

関連情報

- **podman-network** の man ページ

7.5. ポートマッピングによるコンテナー間の通信

2 つのコンテナー間で通信する最も便利な方法は、公開ポートを使用することです。ポートの公開は、自動と手動の 2 つの方法で行うことができます。

手順

1. 未公開のコンテナーを実行します。

```
# podman run -dt --name=web1  ubi9/httpd-24
```

2. 自動的に公開されたコンテナーを実行します。

```
# podman run -dt --name=web2 -P  ubi9/httpd-24
```

3. 手動で公開したコンテナーを実行し、コンテナーポート 80 を公開します。

```
# podman run -dt --name=web3 -p 9090:80  ubi9/httpd-24
```

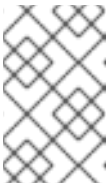
4. すべてのコンテナーを一覧表示します。

```
# podman ps
```

```
CONTAINER ID  IMAGE                                    COMMAND               CREATED
STATUS            PORTS                                     NAMES
f12fa79b8b39  registry.access.redhat.com/ubi9/httpd-24:latest  /usr/bin/run-http...  23
seconds ago  Up 24 seconds ago                                  web1
9024d9e815e2  registry.access.redhat.com/ubi9/httpd-24:latest  /usr/bin/run-http...  13
seconds ago  Up 13 seconds ago  0.0.0.0:43595->8080/tcp, 0.0.0.0:42423->8443/tcp  web2
03bc2a019f1b  registry.access.redhat.com/ubi9/httpd-24:latest  /usr/bin/run-http...  2 seconds
ago   Up 2 seconds ago   0.0.0.0:9090->80/tcp                       web3
```

以下が確認できます。

- コンテナー **web1** には公開ポートがなく、コンテナーネットワークまたはブリッジにでのみアクセスできます。
- コンテナー **web2** は、ポート 43595 と 42423 を自動的にマッピングして、それぞれアプリケーションポート 8080 と 8443 を公開します。



注記

registry.access.redhat.com/9/httpd-24 イメージの Containerfile に **EXPOSE 8080** と **EXPOSE 8443** コマンドがあるため、自動ポートマッピングが可能です。

- コンテナー **web3** は手動でポートを公開しています。ホストポート 9090 は、コンテナーポート 80 にマッピングされます。

5. **web1**、**web3** コンテナーの IP アドレスを表示します。

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' web1
# podman inspect --format='{{.NetworkSettings.IPAddress}}' web3
```

6. <IP>:<port> 表記を使用して **web1** コンテナーに到達します。

```
# curl 10.88.0.14:8080
...
<title>Test Page for the HTTP Server on Red Hat Enterprise Linux</title>
...
```

7. localhost:<port> 表記を使用して **web2** コンテナーに到達します。

```
# curl localhost:43595
...
<title>Test Page for the HTTP Server on Red Hat Enterprise Linux</title>
...
```

8. <IP>:<port> 表記を使用して **web3** コンテナーに到達します:

```
# curl 10.88.0.14:9090
...
<title>Test Page for the HTTP Server on Red Hat Enterprise Linux</title>
...
```

7.6. DNS を利用したコンテナー間の通信

DNS プラグインが有効な場合に、コンテナー名を使用してコンテナーのアドレスを指定します。

前提条件

- DNS プラグインを有効にしたネットワークが **podman network create** コマンドで作成されている。

手順

1. **mynet** ネットワークに接続された **receiver** コンテナーを実行します。

```
# podman run -d --net mynet --name receiver ubi9  sleep 3000
```

2. **sender** コンテナーを実行し、その名前で **receiver** コンテナーにアクセスします。

```
# podman run -it --rm --net mynet --name sender alpine ping receiver

PING rcv01 (10.89.0.2): 56 data bytes
64 bytes from 10.89.0.2: seq=0 ttl=42 time=0.041 ms
64 bytes from 10.89.0.2: seq=1 ttl=42 time=0.125 ms
64 bytes from 10.89.0.2: seq=2 ttl=42 time=0.109 ms
```

CTRL+C で終了します。

sender コンテナーは、**receiver** コンテナーの名前を使用して ping を送信できることがわかります。

7.7. POD 内の 2 つのコンテナー間での通信

同じ Pod 内のコンテナーはすべて、IP アドレス、MAC アドレス、およびポートマッピングを共有します。同じ Pod 内のコンテナー間では、localhost:port の表記で通信が可能です。

手順

1. **web-pod** という名前の Pod を作成します。

```
$ podman pod create --name=web-pod
```

2. **web-container** という名前の Web コンテナーを Pod で実行します。

```
$ podman container run -d --pod web-pod --name=web-container docker.io/library/httpd
```

3. 関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps --pod

CONTAINER ID  IMAGE                         COMMAND           CREATED        STATUS
PORTS       NAMES               POD ID        PODNAME
58653cf0cf09  k8s.gcr.io/pause:3.5                          4 minutes ago  Up 3 minutes ago
4e61a300c194-infra  4e61a300c194  web-pod
b3f4255afdb3  docker.io/library/httpd:latest  httpd-foreground  3 minutes ago  Up 3 minutes
ago               web-container  4e61a300c194  web-pod
```

4. docker.io/library/fedora イメージを元に、**web-pod** でコンテナーを実行します。

```
$ podman container run -it --rm --pod web-pod docker.io/library/fedora curl localhost

<html><body><h1>It works!</h1></body></html>
```

コンテナーが **web-container** に到達できることがわかります。

7.8. POD での通信

Pod 作成時に、Pod 内のコンテナー用のポートを公開する必要があります。

手順

1. **web-pod** という名前の Pod を作成します。

```
# podman pod create --name=web-pod-publish -p 80:80
```

2. すべての Pod を一覧表示します。

```
# podman pod ls

POD ID        NAME         STATUS   CREATED        INFRA ID      # OF CONTAINERS
26fe5de43ab3  publish-pod  Created  5 seconds ago  7de09076d2b3  1
```

3. **web-pod** で **web-container** という名前の Web コンテナーを実行します。

```
# podman container run -d --pod web-pod-publish --name=web-container
docker.io/library/httpd
```

4. コンテナーの一覧表示

```
# podman ps

CONTAINER ID  IMAGE                 COMMAND          CREATED            STATUS
PORTS              NAMES
7de09076d2b3  k8s.gcr.io/pause:3.5                    About a minute ago  Up 23 seconds ago
0.0.0.0:80->80/tcp  26fe5de43ab3-infra
088befb90e59  docker.io/library/httpd  httpd-foreground  23 seconds ago      Up 23 seconds
ago  0.0.0.0:80->80/tcp  web-container
```

5. **web-container** に到達できることを確認します。

```
$ curl localhost:80

<html><body><h1>It works!</h1></body></html>
```

7.9. コンテナーネットワークへの POD のアタッチ

Pod 作成時に Pod 内のコンテナーをネットワークにアタッチします。

手順

1. **pod-net** という名前のネットワークを作成します。

```
# podman network create pod-net

/etc/cni/net.d/pod-net.conflist
```

2. Pod **web-pod** を作成します。

```
# podman pod create --net pod-net --name web-pod
```

3. **web-pod** で **web-container** という名前のコンテナーを実行します。

```
# podman run -d --pod webt-pod --name=web-container docker.io/library/httpd
```

4. オプション。コンテナーが関連付けられている Pod を表示します。

```
# podman ps -p

CONTAINER ID  IMAGE                           COMMAND           CREATED        STATUS
PORTS       NAMES               POD ID        PODNAME
b7d6871d018c  registry.access.redhat.com/ubi9/pause:latest                        9 minutes
ago  Up 6 minutes ago               a8e7360326ba-infra  a8e7360326ba  web-pod
645835585e24  docker.io/library/httpd:latest  httpd-foreground  6 minutes ago  Up 6 minutes
ago               web-container     a8e7360326ba  web-pod
```

検証

- コンテナーに接続されているすべてのネットワークを表示します。

```
# podman ps  --format="{{.Networks}}"

pod-net
```

第8章 コンテナーネットワークモードの設定

この章では、さまざまなネットワークモードの設定方法について説明します。

8.1. 静的 IP でのコンテナー実行

podman run コマンドで **--ip** オプションを指定すると、コンテナーのネットワークインターフェイスが特定の IP アドレスに設定さします (例: 10.88.0.44)。**podman inspect** コマンドを実行して、IP アドレスが正しく設定されていることを確認します。

手順

- コンテナーのネットワークインターフェースを IP アドレス 10.88.0.44 に設定します。

```
# podman run -d --name=myubi --ip=10.88.0.44

registry.access.redhat.com/ubi9/ubi
efde5f0a8c723f70dd5cb5dc3d5039df3b962fae65575b08662e0d5b5f9fbe85
```

検証

- IP アドレスが正しく設定されていることを確認します。

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' myubi

10.88.0.44
```

8.2. SYSTEMD なしでの DHCP プラグイン実行

ユーザー定義のネットワークに接続するには、**Podman run --network** コマンドを使用します。ほとんどのコンテナーイメージには DHCP クライアントがありませんが、**dhcp** プラグインは、コンテナーが DHCP サーバーを操作するためのプロキシー DHCP クライアントとして機能します。

注記

この手順は、ルートフルコンテナーにのみ適用されます。ルートレスコンテナーでは、**dhcp** プラグインは使用されません。

手順

1. **dhcp** プラグインを手動で実行します。

```
# /usr/libexec/cni/dhcp daemon &

[1] 4966
```

2. **dhcp** プラグインが動作していることを確認します。

```
# ps -a | grep dhcp

4966 pts/1    00:00:00 dhcp
```

3. **alpine** コンテナーを実行します。

```
# podman run -it --rm --network=example alpine ip addr show enp1s0

Resolved "alpine" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/alpine:latest...
...
Storing signatures

2: eth0@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP
    link/ether f6:dd:1b:a7:9b:92 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.22/24 brd 192.168.1.255 scope global eth0
    ...
```

この例では、以下のように設定されています。

- **network=example** オプションは、接続するネットワークを example という名前で指定します。
- **alpine** コンテナーで **ip addr show enp1s0** コマンドを実行すると、ネットワークインターフェイス **enp1s0** の IP アドレスを確認します。
- ホストネットワークは 192.168.1.0/24 です。
- **eth0** インターフェイスは、alpine コンテナー用に 192.168.1.122 の IP アドレスをリースしています。

注記

この設定では、有効期限の短いコンテナーとリースの長い DHCP サーバーが多数ある場合に、利用可能な DHCP アドレスが枯渇する可能性があります。

関連情報

- Leasing routable IP addresses with Podman containers

8.3. SYSTEMD を使用した DHCP プラグインの実行

dhcp プラグインを実行するために systemd ユニットファイルを使用できます。

手順

1. ソケットユニットファイルを作成します。

```
# cat /usr/lib/systemd/system/io.podman.dhcp.socket
[Unit]
Description=DHCP Client for CNI

[Socket]
ListenStream=%t/cni/dhcp.sock
SocketMode=0600

[Install]
WantedBy=sockets.target
```

2. サービスユニットファイルを作成します。

```
# cat /usr/lib/systemd/system/io.podman.dhcp.service
[Unit]
Description=DHCP Client CNI Service
Requires=io.podman.dhcp.socket
After=io.podman.dhcp.socket

[Service]
Type=simple
ExecStart=/usr/libexec/cni/dhcp daemon
TimeoutStopSec=30
KillMode=process


[Install]
WantedBy=multi-user.target
Also=io.podman.dhcp.socket
```

3. サービスをすぐに起動します。

```
# systemctl --now enable io.podman.dhcp.socket
```

検証

- ソケットのステータスを確認します。

```
# systemctl status io.podman.dhcp.socket
io.podman.dhcp.socket - DHCP Client for CNI
Loaded: loaded (/usr/lib/systemd/system/io.podman.dhcp.socket; enabled; vendor preset:
disabled)
Active: active (listening) since Mon 2022-01-03 18:08:10 CET; 39s ago
Listen: /run/cni/dhcp.sock (Stream)
CGroup: /system.slice/io.podman.dhcp.socket
```

関連情報

- Leasing routable IP addresses with Podman containers

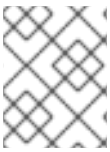
8.4. MACVLAN プラグイン

ほとんどのコンテナーイメージには DHCP クライアントがありませんが、**dhcp** プラグインは、コンテナーが DHCP サーバーを操作するためのプロキシー DHCP クライアントとして機能します。

ホストシステムには、コンテナーへのネットワークアクセス権がありません。ホスト外部からコンテナーへのネットワーク接続を許可するには、コンテナーには、ホストと同じネットワーク上に IP が必要です。**macvlan** プラグインを使用すると、コンテナーをホストと同じネットワークに接続することができます。

注記

この手順は、ルートフルコンテナーにのみ適用されます。ルートレスコンテナーは、**macvlan** および **dhcp** プラグインを使用することができません。



注記

macvlan ネットワークは、**podman network create --macvlan** コマンドで作成することができます。

関連情報

- Leasing routable IP addresses with Podman containers
- **podman-network-create** の man ページ

8.5. ネットワークスタックの CNI から NETAVARK への切り替え

これまで、コンテナーは単一の Container Network Interface (CNI) プラグインに接続されている場合のみ DNS を使用できました。Netavark は、コンテナー用のネットワークスタックです。Netavark は、Podman やその他の OCI(Open Container Initiative) コンテナー管理アプリケーションで使用できます。Podman の高度なネットワークスタックは、Docker の詳細機能にも対応しています。現在、複数のネットワーク上にあるコンテナーは、これらのネットワークにあるコンテナーにアクセスします。

Netavark は以下のことが可能です。

- ブリッジおよび MACVLAN インターフェイスなどのネットワークインターフェイスの作成、管理、削除。
- ネットワークアドレス変換 (NAT) やポートマッピングルールなどのファイアウォールの設定。
- IPv4、IPv6 のサポート。
- 複数のネットワークにおけるコンテナーへの対応改善。

手順

1. **etc/containers/container.conf** ファイルが存在しない場合は、**/usr/share/containers.conf** ファイルを **/etc/containers/** ディレクトリーにコピーしてください。

```
# cp /usr/share/containers/containers.conf /etc/containers/
```

2. **etc/containers/container.conf** ファイルを編集し、**network** セクションに以下の内容を追加します。

```
network_backend="netavark"
```

3. コンテナーや Pod がある場合は、ストレージをリセットして初期状態に戻します。

```
# podman system reset
```

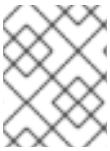
4. システムを再起動します。

```
# reboot
```

検証

- ネットワークスタックが Netavark に変更されていることを確認します。

```
# cat /etc/containers/container.conf
...
[network]
network_backend="netavark"
...
```



注記

Podman 4.0.0 以降をお使いの場合は、**podman info** コマンドでネットワークスタックの設定を確認してください。

関連情報

- Podman 4.0's new network stack:What you need to know
- **podman-system-reset** の man ページ

8.6. NETAVARK から CNI へのネットワークスタックの切り替え

手順

1. **etc/containers/container.conf** ファイルが存在しない場合は、**/usr/share/containers.conf** ファイルを **/etc/containers/** ディレクトリーにコピーしてください。

```
# cp /usr/share/containers/containers.conf /etc/containers/
```

2. **etc/containers/container.conf** ファイルを編集し、**network** セクションに以下の内容を追加します。

```
network_backend="cni"
```

3. コンテナーや Pod がある場合は、ストレージをリセットして初期状態に戻します。

```
# podman system reset
```

4. システムを再起動します。

```
# reboot
```

検証

- ネットワークスタックが CNI に変更されていることを確認します。

```
# cat /etc/containers/container.conf
...
[network]
network_backend="cni"
...
```

注記

Podman 4.0.0 以降をお使いの場合は、**podman info** コマンドでネットワークスタックの設定を確認してください。

関連情報

- Podman 4.0's new network stack:What you need to know
- **podman-system-reset** の man ページ

第9章 PODMAN を使用した OPENSHIFT へのコンテナーの移植

本章では、YAML (YAML Ain't Markup Language) 形式を使用してコンテナーおよび Pod の移植可能な記述を生成する方法を説明します。YAML は、設定データの記述に使用されるテキスト形式です。

YAML ファイルは以下のとおりです。

- 読み取り可能
- 生成が簡単
- 環境間の移植性 (RHEL と OpenShift の間など)
- プログラミング言語間で移植が可能
- 使いやすい (全パラメーターをコマンドラインに追加する必要はない)。

YAML ファイルを使用する理由:

1. 必要最小限の入力でオーケストレーションされたコンテナーおよび Pod のローカルオーケストレーションセットを再実行でき、反復型開発に役立ちます。
2. 同じコンテナーおよび Pod を別のマシンで実行できます。たとえば、OpenShift 環境でアプリケーションを実行し、アプリケーションが正常に機能していることを確認します。**podman generate kube** コマンドを使用して、Kubernetes YAML ファイルを生成できます。次に、**podman play** コマンドを使用して、生成された YAML ファイルを Kubernetes または OpenShift 環境に転送する前に、ローカルシステムで Pod およびコンテナーの作成をテストできます。**podman play** コマンドを使用して、OpenShift または Kubernetes 環境で作成された Pod およびコンテナーを再作成することもできます。

9.1. PODMAN を使用した KUBERNETES YAML ファイルの生成

この手順では、1 つのコンテナーで Pod を作成し、**podman generate kube** コマンドを使用して Kubernetes YAML ファイルを生成する方法を説明します。

前提条件

- Pod が作成されている。詳細は、「Pod の作成」を参照してください。

手順

1. 関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                       COMMAND    CREATED
STATUS                   PORTS  NAMES              POD
5df5c48fea87  registry.access.redhat.com/ubi9/ubi:latest  /bin/bash  Less than a second ago
Up Less than a second ago          myubi              223df6b390b4
3afdcd93de3e  k8s.gcr.io/pause:3.1                                   Less than a second ago  Up Less
than a second ago          223df6b390b4-infra  223df6b390b4
```

2. Pod 名または ID を使用して Kubernetes YAML ファイルを生成します。

```
$ podman generate kube mypod > mypod.yaml
```

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へは反映されないので注意してください。

3. **mypod.yaml** ファイルを表示します。

```
$ cat mypod.yaml
# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-1.6.4
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-06-09T10:31:56Z"
  labels:
app: mypod
  name: mypod
spec:
  containers:
  - command:
        - /bin/bash
        env:
        - name: PATH
              value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
        - name: TERM
              value: xterm
        - name: HOSTNAME
        - name: container
              value: oci
        image: registry.access.redhat.com/ubi9/ubi:latest
        name: myubi
        resources: {}
        securityContext:
              allowPrivilegeEscalation: true
              capabilities: {}
              privileged: false
              readOnlyRootFilesystem: false
        tty: true
        workingDir: /
status: {}
```

関連情報

- **podman-generate-kube** の man ページ
- Podman: Managing pods and containers in a local container runtime

9.2. OPENSHIFT 環境での KUBERNETES YAML ファイルの生成

OpenShift 環境では、**oc create** コマンドを使用して、アプリケーションを記述する YAML ファイルを生成します。

手順

- **myapp** アプリケーションの YAML ファイルを生成します。

```
$ oc create myapp --image=me/myapp:v1 -o yaml --dry-run > myapp.yaml
```

oc create コマンドは **myapp** イメージを作成して実行します。オブジェクトは **--dry-run** オプションを使用して出力され、**myapp.yaml** 出力ファイルにリダイレクトされます。

注記

Kubernetes 環境では、同じフラグを指定して **kubectl create** コマンドを使用できます。

9.3. PODMAN でのコンテナーおよび POD の起動

生成された YAML ファイルを使用すると、任意の環境でコンテナーおよび Pod を自動的に起動できます。YAML ファイルは Podman では生成されないことに注意してください。**podman play kube** コマンドを使用すると、YAML 入力ファイルに基づいて Pod およびコンテナーを再作成できます。

手順

1. **mypod.yaml** ファイルから Pod およびコンテナーを作成します。

```
$ podman play kube mypod.yaml
Pod:
b8c5b99ba846ccff76c3ef257e5761c2d8a5ca4d7ffa3880531aec79c0dacb22
Container:
848179395ebd33dd91d14ffbde7ae273158d9695a081468f487af4e356888ece
```

2. すべての Pod を一覧表示します。

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED         # OF CONTAINERS   INFRA ID
b8c5b99ba846  mypod   Running   19 seconds ago  2                 aa4220eaf4bb
```

3. 関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                 COMMAND    CREATED            STATUS
PORTS  NAMES              POD
848179395ebd  registry.access.redhat.com/ubi9/ubi:latest  /bin/bash  About a minute ago  Up
About a minute ago        myubi             b8c5b99ba846
aa4220eaf4bb  k8s.gcr.io/pause:3.1                        About a minute ago  Up About a
minute ago        b8c5b99ba846-infra  b8c5b99ba846
```

podman ps コマンドからの Pod ID と、 **podman pod ps** コマンドからの Pod ID が一致します。

関連情報

- **podman-play-kube** の man ページ
- Podman can now ease the transition to Kubernetes and CRI-O

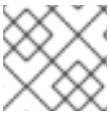
9.4. OPENSHIFT 環境でのコンテナーおよび POD の起動

oc create コマンドを使用して、OpenShift 環境で Pod およびコンテナーを作成できます。

手順

- OpenShift 環境で YAML ファイルから Pod を作成します。

```
$ oc create -f mypod.yaml
```



注記

Kubernetes 環境では、同じフラグを指定して **kubectl create** コマンドを使用できます。

9.5. PODMAN を使用したコンテナーと POD の手動実行

以下の手順は、Podman を使用して MariaDB データベースと対になる WordPress コンテンツマネジメントシステムを手動で作成する方法を説明します。

次のようなディレクトリーレイアウトがあるとします。

```
├── mariadb-conf
│   ├── Containerfile
│   ├── my.cnf
```

手順

1. **mariadb-conf/Containerfile** ファイルを表示します。

```
$ cat mariadb-conf/Containerfile
FROM docker.io/library/mariadb
COPY my.cnf /etc/mysql/my.cnf
```

2. **mariadb-conf/my.cnf** ファイルを表示します。

```
[client-server]
# Port or socket location where to connect
port = 3306
socket = /run/mysqld/mysqld.sock

# Import all .cnf files from the configuration directory
[mariadbd]
skip-host-cache
skip-name-resolve
bind-address = 127.0.0.1

!includedir /etc/mysql/mariadb.conf.d/
!includedir /etc/mysql/conf.d/
```

3. **mariadb-conf/Containerfile** を使って **docker.io/library/mariadb** イメージをビルドします。

```
$ cd  mariadb-conf
$ podman build -t mariadb-conf .
$ cd ..
STEP 1: FROM docker.io/library/mariadb
```

```
Trying to pull docker.io/library/mariadb:latest...
Getting image source signatures
Copying blob 7b1a6ab2e44d done
...
Storing signatures
STEP 2: COPY my.cnf /etc/mysql/my.cnf
STEP 3: COMMIT mariadb-conf
--> ffae584aa6e
Successfully tagged localhost/mariadb-conf:latest
ffae584aa6e733ee1cdf89c053337502e1089d1620ff05680b6818a96eec3c17
```

4. オプション。すべてのイメージを一覧表示します。

```
$ podman images
LIST IMAGES
REPOSITORY                                    TAG         IMAGE ID      CREATED
SIZE
localhost/mariadb-conf                        latest      b66fa0fa0ef2  57 seconds ago
416 MB
```

5. **wordpresspod** という名前の pod を作成し、コンテナーとホストシステム間のポートマッピングを設定します。

```
$ podman pod create --name wordpresspod -p 8080:80
```

6. **wordpresspod** pod の中に **mydb** コンテナーを作成します。

```
$ podman run --detach --pod wordpresspod \
   -e MYSQL_ROOT_PASSWORD=1234 \
   -e MYSQL_DATABASE=mywpdb \
   -e MYSQL_USER=mywpuser \
   -e MYSQL_PASSWORD=1234 \
   --name mydb localhost/mariadb-conf
```

7. **wordpresspod** pod の中に **myweb** コンテナーを作成します。

```
$ podman run --detach --pod wordpresspod \
   -e WORDPRESS_DB_HOST=127.0.0.1 \
   -e WORDPRESS_DB_NAME=mywpdb \
   -e WORDPRESS_DB_USER=mywpuser \
   -e WORDPRESS_DB_PASSWORD=1234 \
   --name myweb docker.io/wordpress
```

8. オプション。関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps --pod -a
CONTAINER ID  IMAGE                          COMMAND             CREATED
STATUS                  PORTS               NAMES               POD ID        PODNAME
9ea56f771915  k8s.gcr.io/pause:3.5                               Less than a second ago  Up Less
than a second ago  0.0.0.0:8080->80/tcp  4b7f054a6f01-infra  4b7f054a6f01  wordpresspod
60e8dbbabac5  localhost/mariadb-conf:latest       mariadbd            Less than a second ago
Up Less than a second ago  0.0.0.0:8080->80/tcp  mydb                4b7f054a6f01
wordpresspod
```

```
045d3d506e50  docker.io/library/wordpress:latest  apache2-foregroun...  Less than a second ago  Up Less than a second ago  0.0.0.0:8080->80/tcp  myweb                4b7f054a6f01  wordpresspod
```

検証

- Pod が実行されていることを確認します。http://localhost:8080/wp-admin/install.php のページにアクセスするか、**curl** コマンドを使用してください。

```
$ curl  http://localhost:8080/wp-admin/install.php
<!DOCTYPE html>
<html xml:lang="ja-JP">
<head>
...
</head>
<body class="wp-core-ui">
<p id="logo">WordPress</p>
    <h1>Welcome</h1>
...
```

関連情報

- Build Kubernetes pods with Podman play kube
- **podman-play-kube** の man ページ

9.6. PODMAN を使用した YAML ファイルの生成

Kubernetes の YAML ファイルは、**podman generate kube** コマンドで生成できます。

前提条件

- **wordpresspod** という名前の Pod が作成されている。詳細は、「Pod の作成」を参照してください。

手順

1. 関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps --pod -a
CONTAINER ID  IMAGE                           COMMAND               CREATED
STATUS                   PORTS               NAMES               POD ID        PODNAME
9ea56f771915  k8s.gcr.io/pause:3.5                                  Less than a second ago  Up Less than a second ago  0.0.0.0:8080->80/tcp  4b7f054a6f01-infra  4b7f054a6f01  wordpresspod
60e8dbbabac5  localhost/mariadb-conf:latest       mariadbd              Less than a second ago  Up Less than a second ago  0.0.0.0:8080->80/tcp  mydb                4b7f054a6f01  wordpresspod
045d3d506e50  docker.io/library/wordpress:latest  apache2-foregroun...  Less than a second ago  Up Less than a second ago  0.0.0.0:8080->80/tcp  myweb               4b7f054a6f01  wordpresspod
```

2. Pod 名または ID を使用して Kubernetes YAML ファイルを生成します。

```
$ podman generate kube wordpresspod >> wordpresspod.yaml
```

検証

- **wordpresspod.yaml** ファイルを表示します。

```
$ cat wordpresspod.yaml
...
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-12-09T15:09:30Z"
  labels:
    app: wordpresspod
  name: wordpresspod
spec:
  containers:
  - args:
        value: podman
      - name: MYSQL_PASSWORD
        value: "1234"
      - name: MYSQL_MAJOR
        value: "8.0"
      - name: MYSQL_VERSION
        value: 8.0.27-1debian10
      - name: MYSQL_ROOT_PASSWORD
        value: "1234"
      - name: MYSQL_DATABASE
        value: mywpdb
      - name: MYSQL_USER
        value: mywpuser
        image: mariadb
            name: mydb
            ports:
            - containerPort: 80
              hostPort: 8080
              protocol: TCP
  - args:
    - name: WORDPRESS_DB_NAME
      value: mywpdb
    - name: WORDPRESS_DB_PASSWORD
      value: "1234"
    - name: WORDPRESS_DB_HOST
      value: 127.0.0.1
    - name: WORDPRESS_DB_USER
      value: mywpuser
      image: docker.io/library/wordpress:latest
      name: myweb
```

関連情報

- Build Kubernetes pods with Podman play kube
- **podman-play-kube** の man ページ

9.7. PODMAN を使用したコンテナーと POD の自動実行

次に、**podman play kube** コマンドを使用して、生成された YAML ファイルを Kubernetes または OpenShift 環境に転送する前に、ローカルシステムで Pod およびコンテナーの作成をテストできます。

podman play kube コマンドは、docker compose コマンドと同様に YAML ファイルを使って、コンテナーが複数ある Pod を自動的に構築して実行することも可能です。以下の条件が該当する場合には、自動的にイメージが構築されます。

1. YAML ファイルで使用されているイメージと同じ名前のディレクトリーが存在する
2. そのディレクトリーには Containerfile が含まれている

前提条件

- **wordpresspod** という名前の Pod が作成されている。詳細は、Podman を使用したコンテナーと Pod の手動実行 を参照してください。
- YAML ファイルが生成されている。詳細は、Podman を使用した YAML ファイルの生成のセクションをご覧ください。
- 最初からやり直す場合は、ローカルに保存されているイメージを削除してください。

```
$ podman rmi localhost/mariadb-conf
$ podman rmi docker.io/library/wordpress
$ podman rmi docker.io/library/mysql
```

手順

1. **wordpress.yaml** ファイルを使用して、wordpress Pod を作成します。

```
STEP 1/2: FROM docker.io/library/mariadb
STEP 2/2: COPY my.cnf /etc/mysql/my.cnf
COMMIT localhost/mariadb-conf:latest
--> 428832c45d0
Successfully tagged localhost/mariadb-conf:latest
428832c45d07d78bb9cb34e0296a7dc205026c2fe4d636c54912c3d6bab7f399
Trying to pull docker.io/library/wordpress:latest...
Getting image source signatures
Copying blob 99c3c1c4d556 done
...
Storing signatures
Pod:
3e391d091d190756e655219a34de55583eed3ef59470aadd214c1fc48cae92ac
Containers:
6c59ebe968467d7fdb961c74a175c88cb5257fed7fb3d375c002899ea855ae1f
29717878452ff56299531f79832723d3a620a403f4a996090ea987233df0bc3d
```

podman play kube コマンド:

- **docker.io/library/mariadb** イメージを元に **localhost/mariadb-conf:latest** イメージを自動構築します。
- **docker.io/library/wordpress:latest** のイメージをプルします。

- **wordpresspod-mydb** と **wordpresspod-myweb** の 2 つのコンテナーが含まれる、**wordpresspod** という名前の Pod を作成します。

2. すべてのコンテナーと Pod を表示します。

```
$ podman ps --pod -a
CONTAINER ID  IMAGE                                COMMAND               CREATED         STATUS
PORTS                   NAMES               POD ID        PODNAME
a1dbf7b5606c  k8s.gcr.io/pause:3.5                                      3 minutes ago   Up 2 minutes ago
0.0.0.0:8080->80/tcp  3e391d091d19-infra  3e391d091d19  wordpresspod
6c59ebe96846  localhost/mariadb-conf:latest        mariadbd              2 minutes ago   Exited (1)
2 minutes ago  0.0.0.0:8080->80/tcp  wordpresspod-mydb   3e391d091d19  wordpresspod
29717878452f  docker.io/library/wordpress:latest  apache2-foregroun...  2 minutes ago   Up 2
minutes ago          0.0.0.0:8080->80/tcp  wordpresspod-myweb  3e391d091d19
wordpresspod
```

検証

- Pod が実行されていることを確認します。http://localhost:8080/wp-admin/install.php のページにアクセスするか、**curl** コマンドを使用してください。

```
$ curl  http://localhost:8080/wp-admin/install.php
<!DOCTYPE html>
<html xml:lang="ja-JP">
<head>
...
</head>
<body class="wp-core-ui">
<p id="logo">WordPress</p>
    <h1>Welcome</h1>
...
```

関連情報

- Build Kubernetes pods with Podman play kube
- **podman-play-kube** の man ページ

9.8. PODMAN を使用した POD の自動停止/削除

podman play kube --down コマンドは、すべての Pod とそのコンテナーを停止して削除します。

注記

ボリュームが使用中の場合には削除されません。

前提条件

- **wordpresspod** という名前の Pod が作成されている。詳細は、Podman を使用したコンテナーと Pod の手動実行 を参照してください。
- YAML ファイルが生成されている。詳細は、Podman を使用した YAML ファイルの生成のセクションをご覧ください。

- Pod が実行中である。詳細は、Podman を使用したコンテナーや Pod の自動実行 を参照してください。

手順

- **wordpresspod.yaml** ファイルで作成された全 Pod とコンテナーを削除します。

```
$ podman play kube --down wordpresspod.yaml
Pods stopped:
3e391d091d190756e655219a34de55583eed3ef59470aadd214c1fc48cae92ac
Pods removed:
3e391d091d190756e655219a34de55583eed3ef59470aadd214c1fc48cae92ac
```

検証

- **wordpresspod.yaml** ファイルで作成された全 Pod とコンテナーが削除されたことを確認します。

```
$ podman ps --pod -a
CONTAINER ID  IMAGE                       COMMAND           CREATED
STATUS                  PORTS             NAMES             POD ID        PODNAME
```

関連情報

- Build Kubernetes pods with Podman play kube
- **podman-play-kube** の man ページ

第10章 PODMAN を使用した SYSTEMD へのコンテナーの移植

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。

Podman は、当初、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。systemd 初期化サービスを使用して、Pod およびコンテナーを操作できます。**podman generate systemd** コマンドを使用して、コンテナーおよび Pod の systemd ユニットファイルを生成できます。

systemd ユニットファイルを使用すると、以下が可能になります。

- コンテナーまたは Pod を systemd サービスとして起動するように設定します。
- コンテナー化されたサービスを実行して依存関係を確認する順序を定義します (たとえば、別のサービスが実行されていること、ファイルが利用可能であること、またはリソースがマウントされていることなど)。
- **systemctl** コマンドを使用して、systemd システムの状態を制御します。

本章では、systemd ユニットファイルを使用してコンテナーおよび Pod の移植可能な記述を生成する方法を説明します。

10.1. SYSTEMD サービスの有効化

サービスの有効化には、さまざまなオプションがあります。

手順

- サービスの有効化:
  - システムの起動時にサービスを有効にするには、ユーザーがログインしているかどうかに拘らず、次のコマンドを実行します。

    ```
    # systemctl enable <service>
    ```

    systemd ユニットファイルを **/etc/systemd/system** ディレクトリーにコピーする必要があります。
  - ユーザーのログイン時にサービスを起動し、ユーザーのログアウト時に停止するには、次のコマンドを実行します。

    ```
    $ systemctl --user enable <service>
    ```

    systemd ユニットファイルを **$HOME/.config/systemd/user** ディレクトリーにコピーする必要があります。
  - システムの起動時にサービスを起動し、ログアウト後もそのまま起動した状態を保つには、次のコマンドを実行します。

    ```
    # loginctl enable-linger <username>
    ```

関連情報

- **systemctl** の man ページ
- **loginctl** の man ページ
- システム起動時に systemd サービスの開始

10.2. PODMAN を使用した SYSTEMD ユニットファイルの生成

Podman を使用することで、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既存のコンテナーと Pod の systemd ユニットファイルを生成できます。生成されたユニットファイルは頻繁に変更され (Podman に対する更新)、**podman generate systemd** で最新のユニットファイルの取得を確認できるので、**podman generate systemd** の使用を推奨します。

手順

1. コンテナー (例: **myubi**) を作成します。

```
$ podman create --name myubi registry.access.redhat.com/ubi9:latest sleep infinity
0280afe98bb75a5c5e713b28de4b7c5cb49f156f1cce4a208f13fee2f75cb453
```

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-myubi.service** ファイルに送ります。

```
$ podman generate systemd --name myubi > ~/.config/systemd/user/container-myubi.service
```

検証手順

- 生成された systemd ユニットファイルの内容を表示します。

```
$ cat ~/.config/systemd/user/container-myubi.service
# container-myubi.service
# autogenerated by Podman 3.3.1
# Wed Sep  8 20:34:46 CEST 2021

[Unit]
Description=Podman container-myubi.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=/run/user/1000/containers

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman start myubi
ExecStop=/usr/bin/podman stop -t 10 myubi
ExecStopPost=/usr/bin/podman stop -t 10 myubi
PIDFile=/run/user/1000/containers/overlay-
containers/9683103f58a32192c84801f0be93446cb33c1ee7d9cdda225b78049d7c5deea4/user
data/conmon.pid
```

```
Type=forking

[Install]
WantedBy=multi-user.target default.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。

関連情報

- Running containers with Podman and shareable systemd services

10.3. PODMAN を使用した SYSTEMD ユニットファイルの自動生成

デフォルトで、Podman は既存のコンテナーまたは Pod のユニットファイルを生成します。**podman generate systemd --new** を使用して、移植可能な別の systemd ユニットファイルを生成できます。**--new** フラグでは、コンテナーの作成、起動、削除を行うユニットファイルを生成するように Podman に指示します。

手順

1. システムで使用するイメージをプルします。たとえば、**httpd-24** イメージをプルするには、以下を実行します。

```
# podman pull registry.access.redhat.com/ubi9/httpd-24
```

2. オプション。システムで利用可能なイメージの一覧を表示します。

```
# podman images
REPOSITORY                              TAG         IMAGE ID      CREATED       SIZE
registry.access.redhat.com/ubi9/httpd-24  latest      8594be0a0b57  2 weeks ago   462
MB
```

3. **httpd** コンテナーを作成します。

```
# podman create --name httpd -p 8080:8080 registry.access.redhat.com/ubi9/httpd-24
cdb9f981cf143021b1679599d860026b13a77187f75e46cc0eac85293710a4b1
```

4. オプション。コンテナーが作成されたことを確認します。

```
# podman ps -a
CONTAINER ID  IMAGE                                COMMAND             CREATED
STATUS      PORTS                 NAMES
cdb9f981cf14  registry.access.redhat.com/ubi9/httpd-24:latest  /usr/bin/run-http...  5 minutes
ago  Created     0.0.0.0:8080->8080/tcp  httpd
```

5. **httpd** コンテナーの systemd ユニットファイルを生成します。

```
# podman generate systemd --new --files --name httpd
/root/container-httpd.service
```

6. 生成された **container-httpd.service** systemd ユニットファイルの内容を表示します。

```
# cat /root/container-httpd.service
# container-httpd.service
# autogenerated by Podman 3.3.1
# Wed Sep  8 20:41:44 CEST 2021

[Unit]
Description=Podman container-httpd.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=%t/containers

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStartPre=/bin/rm -f %t/%n.ctr-id
ExecStart=/usr/bin/podman run --cidfile=%t/%n.ctr-id --sdnotify=conmon --cgroups=no-
conmon --rm -d --replace --name httpd -p 8080:8080 registry.access.redhat.com/ubi9/httpd-
24
ExecStop=/usr/bin/podman stop --ignore --cidfile=%t/%n.ctr-id
ExecStopPost=/usr/bin/podman rm -f --ignore --cidfile=%t/%n.ctr-id
Type=notify
NotifyAccess=all

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

注記

--new オプションを使用してユニットファイルを生成した場合には、コンテナーと Pod の存在は想定されていません。したがって、サービスの起動時に (**ExecStart** の行を参照)、**podman start** コマンドではなく、**podman run** コマンドを実行します。たとえば、「Generating a systemd unit file using Podman 」のセクションを参照してください。

- **podman run** コマンドは、以下のコマンドラインオプションを使用します。
  - **--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するので、systemd は正しいサービスステータスを報告して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。

7. **/usr/lib/systemd/system** にユニットファイルをコピーして root ユーザーとしてインストールします。

```
# cp -Z container-httpd.service  /etc/systemd/system
```

8. **container-httpd.service** を有効にして起動します。

```
# systemctl daemon-reload
# systemctl enable --now container-httpd.service
Created symlink /etc/systemd/system/multi-user.target.wants/container-httpd.service →
/etc/systemd/system/container-httpd.service.
Created symlink /etc/systemd/system/default.target.wants/container-httpd.service →
/etc/systemd/system/container-httpd.service.
```

検証手順

- **container-httpd.service** のステータスを確認します。

```
# systemctl status container-httpd.service
   ● container-httpd.service - Podman container-httpd.service
      Loaded: loaded (/etc/systemd/system/container-httpd.service; enabled; vendor preset:
disabled)
      Active: active (running) since Tue 2021-08-24 09:53:40 EDT; 1min 5s ago
        Docs: man:podman-generate-systemd(1)
     Process: 493317 ExecStart=/usr/bin/podman run --conmon-pidfile /run/container-
httpd.pid --cidfile /run/container-httpd.ctr-id --cgroups=no-conmon -d --repla>
     Process: 493315 ExecStartPre=/bin/rm -f /run/container-httpd.pid /run/container-httpd.ctr-
id (code=exited, status=0/SUCCESS)
    Main PID: 493435 (conmon)
   ...
```

関連情報

- Systemd Integration with Podman 2.0
- システム起動時に systemd サービスの開始

10.4. SYSTEMD を使用したコンテナーの自動起動

systemctl コマンドを使用して、systemd システムおよびサービスマネージャーの状態を制御できます。本セクションでは、root 以外のユーザーでサービスの有効化、起動、停止を行う方法を説明します。root ユーザーとしてサービスをインストールするには、**--user** オプションを省略します。

手順

1. 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
```

2. サービス **container.service** を有効にし、起動時に開始します。

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

3. サービスをすぐに起動します。

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

4. サービスの状況を表示するには、次のコマンドを実行します。

```
$ systemctl --user status container.service
● container.service - Podman container.service
   Loaded: loaded (/home/user/.config/systemd/user/container.service; enabled; vendor
preset: enabled)
   Active: active (running) since Wed 2020-09-16 11:56:57 CEST; 8s ago
     Docs: man:podman-generate-systemd(1)
  Process: 80602 ExecStart=/usr/bin/podman run --conmon-pidfile
//run/user/1000/container.service-pid --cidfile //run/user/1000/container.service-cid -d ubi9-
minimal:>
  Process: 80601 ExecStartPre=/usr/bin/rm -f //run/user/1000/container.service-pid
//run/user/1000/container.service-cid (code=exited, status=0/SUCCESS)
 Main PID: 80617 (conmon)
   CGroup: /user.slice/user-1000.slice/user@1000.service/container.service
           ├─ 2870 /usr/bin/podman
           ├─80612 /usr/bin/slirp4netns --disable-host-loopback --mtu 65520 --enable-sandbox -
-enable-seccomp -c -e 3 -r 4 --netns-type=path /run/user/1000/netns/cni->
           ├─80614 /usr/bin/fuse-overlayfs -o
lowerdir=/home/user/.local/share/containers/storage/overlay/l/YJSPGXM2OCDZPLMLXJOW3N
RF6Q:/home/user/.local/share/contain>
           ├─80617 /usr/bin/conmon --api-version 1 -c
cbc75d6031508dfd3d78a74a03e4ace1732b51223e72a2ce4aa3bfe10a78e4fa -u
cbc75d6031508dfd3d78a74a03e4ace1732b51223e72>
           └─cbc75d6031508dfd3d78a74a03e4ace1732b51223e72a2ce4aa3bfe10a78e4fa
             └─80626 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1d
```

systemctl is-enabled container.service コマンドを使用して、サービスが有効であるかどうかを確認できます。

検証手順

- 実行中または終了したコンテナーを一覧表示します。

```
# podman ps
CONTAINER ID  IMAGE                                  COMMAND  CREATED         STATUS
PORTS  NAMES
f20988d59920  registry.access.redhat.com/ubi9-minimal:latest  top      12 seconds ago  Up 11
seconds ago         funny_zhukovsky
```

注記

container.service を停止するには、以下を入力します。

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

関連情報

- **systemctl** の man ページ
- Running containers with Podman and shareable systemd services

- systemd によるサービス管理

10.5. SYSTEMD を使用した POD の自動起動

複数のコンテナーを systemd サービスとして起動できます。**systemctl** コマンドは、Pod でだけ使用するようにしてください。コンテナーは Pod サービスと内部の infra-container で管理されているので **systemctl** を使用して個別にコンテナーを開始または停止しないでください。

手順

1. たとえば、**systemd-pod** などの空の Pod を作成します。

```
$ podman pod create --name systemd-pod
11d4646ba41b1fffa51c108cbdf97cfab3213f7bd9b3e1ca52fe81b90fed5577
```

2. オプション。すべての Pod を一覧表示します。

```
$ podman pod ps
POD ID        NAME         STATUS   CREATED         # OF CONTAINERS  INFRA ID
11d4646ba41b  systemd-pod  Created  40 seconds ago  1                8a428b257111
11d4646ba41b1fffa51c108cbdf97cfab3213f7bd9b3e1ca52fe81b90fed5577
```

3. 空の Pod に 2 つのコンテナーを作成します。たとえば、**container0** と **container1** を **systemd-pod** に作成するには、以下を実行します。

```
$ podman create --pod systemd-pod --name container0 registry.access.redhat.com/ubi9 top
$ podman create --pod systemd-pod --name container1 registry.access.redhat.com/ubi9 top
```

4. オプション。関連付けられている全 Pod およびコンテナーを一覧表示します。

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                COMMAND  CREATED        STATUS
PORTS   NAMES              POD ID        PODNAME
24666f47d9b2  registry.access.redhat.com/ubi9:latest  top      3 minutes ago  Created
container0          3130f724e229  systemd-pod
56eb1bf0cdfe  k8s.gcr.io/pause:3.2                          4 minutes ago  Created
3130f724e229-infra  3130f724e229  systemd-pod
62118d170e43  registry.access.redhat.com/ubi9:latest  top      3 seconds ago  Created
container1          3130f724e229  systemd-pod
```

5. 新規 Pod の systemd ユニットファイルを生成します。

```
$ podman generate systemd --files --name systemd-pod
/home/user1/pod-systemd-pod.service
/home/user1/container-container0.service
/home/user1/container-container1.service
```

systemd ユニットファイルは、3 つ作成される点に注意してください。1 つは **systemd-pod** の Pod 向け、2 つは **container0** と **container1** のコンテナー向けです。

6. **pod-systemd-pod.service** ユニットファイルを表示します。

```
$ cat pod-systemd-pod.service
```

```
# pod-systemd-pod.service
# autogenerated by Podman 3.3.1
# Wed Sep  8 20:49:17 CEST 2021

[Unit]
Description=Podman pod-systemd-pod.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=
Requires=container-container0.service container-container1.service
Before=container-container0.service container-container1.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman start bcb128965b8e-infra
ExecStop=/usr/bin/podman stop -t 10 bcb128965b8e-infra
ExecStopPost=/usr/bin/podman stop -t 10 bcb128965b8e-infra
PIDFile=/run/user/1000/containers/overlay-
containers/1dfdcf20e35043939ea3f80f002c65c00d560e47223685dbc3230e26fe001b29/userda
ta/conmon.pid
Type=forking

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

- **[Unit]** セクションの **Requires** 行は、**container-container0.service** と **container-container1.service** ユニットファイルの依存関係を定義します。両方のユニットファイルがアクティベートされます。

- **[Service]** セクションの **ExecStart** 行および **ExecStop** 行はそれぞれ infra-container を開始して停止します。

7. **container-container0.service** ユニットファイルを表示します。

```
$ cat container-container0.service
# container-container0.service
# autogenerated by Podman 3.3.1
# Wed Sep  8 20:49:17 CEST 2021

[Unit]
Description=Podman container-container0.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=/run/user/1000/containers
BindsTo=pod-systemd-pod.service
After=pod-systemd-pod.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman start container0
```

```
ExecStop=/usr/bin/podman stop -t 10 container0
ExecStopPost=/usr/bin/podman stop -t 10 container0
PIDFile=/run/user/1000/containers/overlay-
containers/4bccd7c8616ae5909b05317df4066fa90a64a067375af5996fdef9152f6d51f5/userdat
a/conmon.pid
Type=forking

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

- **[Unit]** セクションの **BindsTo** l行は、**pod-systemd-pod.service** ユニットファイルの依存関係を定義します。
- **[Service]** セクションの **ExecStart** 行および **ExecStop** 行では、それぞれ **container0** を起動および停止します。

8. **container-container1.service** ユニットファイルを表示します。

```
$ cat container-container1.service
```

9. 生成されたファイルをすべて **$HOME/.config/systemd/user** にコピーして、root 以外のユーザーとしてインストールします。

```
$ cp pod-systemd-pod.service container-container0.service container-container1.service
$HOME/.config/systemd/user
```

10. ユーザーのログイン時に、サービスを有効にして開始します。

```
$ systemctl enable --user pod-systemd-pod.service
Created symlink /home/user1/.config/systemd/user/multi-user.target.wants/pod-systemd-
pod.service → /home/user1/.config/systemd/user/pod-systemd-pod.service.
Created symlink /home/user1/.config/systemd/user/default.target.wants/pod-systemd-
pod.service → /home/user1/.config/systemd/user/pod-systemd-pod.service.
```

サービスは、ユーザーのログアウト時に停止される点に注意してください。

検証手順

- サービスが有効になっているかどうかを確認します。

```
$ systemctl is-enabled pod-systemd-pod.service
enabled
```

関連情報

- **podman-create** の man ページ
- **podman-generate-systemd** の man ページ
- **systemctl** の man ページ
- Running containers with Podman and shareable systemd services
- システム起動時に systemd サービスの開始

10.6. PODMAN を使用したコンテナーの自動更新

podman auto-update コマンドを使用すると、自動更新のポリシーに従ってコンテナーを自動的に更新できます。**podman auto-update** コマンドは、コンテナーイメージがレジストリーで更新されるとサービスを更新します。自動更新を使用するには、 **--label "io.containers.autoupdate=image"** ラベルで作成され、**podman generate systemd --new** コマンドで生成される systemd ユニットでコンテナーを実行する必要があります。

Podman は、**"io.containers.autoupdate"** ラベルが **"image"** に設定されている実行中のコンテナーを検索して、コンテナーのレジストリーと通信します。イメージが変更されると、Podman は対応する systemd ユニットを再起動して古いコンテナーを停止し、新しいイメージで新規のコンテナーを作成します。そのため、コンテナー、その環境、およびすべての依存関係が再起動されます。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

1. **registry.access.redhat.com/ubi9/ubi-init** イメージをもとに、**myubi** コンテナーを起動します。

```
# podman run --label "io.containers.autoupdate=image" \
--name myubi  -dt registry.access.redhat.com/ubi9/ubi-init top
bc219740a210455fa27deacc96d50a9e20516492f1417507c13ce1533dbdcd9d
```

2. オプション:実行中または終了したコンテナーを一覧表示します。

```
# podman ps -a
CONTAINER ID  IMAGE                                COMMAND  CREATED        STATUS
PORTS   NAMES
76465a5e2933  registry.access.redhat.com/9/ubi-init:latest  top      24 seconds ago  Up 23
seconds ago          myubi
```

3. **myubi** コンテナーの systemd ユニットファイルを生成します。

```
# podman generate systemd --new --files --name myubi
/root/container-myubi.service
```

4. **/usr/lib/systemd/system** にユニットファイルをコピーして root ユーザーとしてインストールします。

```
# cp -Z ~/container-myubi.service /usr/lib/systemd/system
```

5. 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 daemon-reload
```

6. コンテナーを起動して、ステータスを確認します。

```
# systemctl start container-myubi.service
# systemctl status container-myubi.service
```

7. コンテナーを自動更新します。

```
# podman auto-update
```

関連情報

- Systemd Integration with Podman 2.0
- Running containers with Podman and shareable systemd services
- システム起動時に systemd サービスの開始

10.7. SYSTEMD を使用したコンテナーの自動更新

「Podman を使用したコンテナーの自動更新」のセクションで説明したように、**podman auto-update** コマンドを使用してコンテナーを更新できます。カスタムスクリプトに組み込み、必要に応じて呼び出すことができます。コンテナーを自動更新する別の方法として、プレインストールされた **podman-auto-update.timer** および **podman-auto-update.service** systemd サービスを使用できます。**podman-auto-update.timer** は、特定の日付と時刻で自動更新をトリガーするように設定できます。**podman-auto-update.service** はさらに **systemctl** コマンドから起動することも、他の systemd サービスの依存関係として使用することもできます。その結果、時間およびイベントに基づく自動更新は、個々のニーズやユースケースを満たすためにさまざまな方法でトリガーできます。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

1. **podman-auto-update.service** ユニットファイルを表示します。

```
# cat /usr/lib/systemd/system/podman-auto-update.service

[Unit]
Description=Podman auto-update service
Documentation=man:podman-auto-update(1)
Wants=network.target
After=network-online.target

[Service]
Type=oneshot
ExecStart=/usr/bin/podman auto-update

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

2. **podman-auto-update.timer** ユニットファイルを表示します。

```
# cat /usr/lib/systemd/system/podman-auto-update.timer

[Unit]
Description=Podman auto-update timer

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

この例では、**podman auto-update** コマンドは、毎日、深夜に起動します。

3. システム起動時に **podman-auto-update.timer** サービスを有効にします。

```
# systemctl enable podman-auto-update.timer
```

4. systemd サービスを起動します。

```
# systemctl start podman-auto-update.timer
```

5. オプション:タイマーを一覧表示します。

```
# systemctl list-timers --all
NEXT                         LEFT      LAST                         PASSED       UNIT
ACTIVATES
Wed 2020-12-09 00:00:00 CET  9h left   n/a                          n/a          podman-auto-
update.timer     podman-auto-update.service
```

podman-auto-update.timer が **podman-auto-update.service** を有効化したことが確認できます。

関連情報

- Systemd Integration with Podman 2.0
- Running containers with Podman and shareable systemd services
- システム起動時に systemd サービスの開始

第11章 実行中の UBI コンテナーへのソフトウェアの追加

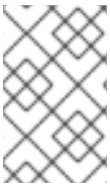
UBI (Red Hat Universal Base Images) は、RHEL コンテンツのサブセットから構築されます。UBI は、UBI でインストールするために自由に利用可能な RHEL パッケージのサブセットも提供します。実行中のコンテナーにソフトウェアを追加または更新するには、RPM パッケージおよび更新を含む dnf リポジトリーを使用します。UBI は、Python、Perl、Node.js、Ruby などの事前にビルドされた言語ランタイムコンテナーイメージを提供します。

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 init および UBI 標準イメージでは、**dnf** コマンドを使用します。
- UBI の最小イメージでは、**microdnf** コマンドを使用します。

注記

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と、パッケージを一時的に追加します。この変更はコンテナーイメージに保存されません。パッケージの変更を永続化するには、「Buildah を使用した Containerfile からのイメージのビルド」セクションを参照してください。



注記

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 を更新する手順が異なります。

11.1. 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、RHEL Base リポジトリーおよび AppStream 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。

関連情報

- Universal Base Images (UBI): Images, repositories, packages, and source code を参照してください。

11.2. 標準の UBI コンテナーへのソフトウェアの追加

標準の UBI コンテナー内にソフトウェアを追加するには、UBI 以外の dnf リポジトリーを無効にして、ビルドするコンテナーが再配布されるようにします。

手順

1. **registry.access.redhat.com/ubi9/ubi** イメージをプルして実行します。

```
$ podman run -it --name myubi registry.access.redhat.com/ubi9/ubi
```

2. **myubi** コンテナーにパッケージを追加します。

- UBI リポジトリーにあるパッケージを追加するには、UBI リポジトリー以外の dnf リポジトリーをすべて無効にします。たとえば、**bzip2** パッケージを追加するには、次のコマンドを実行します。

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

- UBI リポジトリーにないパッケージを追加するには、リポジトリーを無効にしないでください。たとえば、**zsh** パッケージを追加するには、次のコマンドを実行します。

```
# dnf install zsh
```

- 別のホストリポジトリーにあるパッケージを追加するには、必要なリポジトリーを明示的に有効にします。たとえば、**codeready-builder-for-rhel-8-x86_64-rpms** リポジトリーから **python38-devel** パッケージをインストールするには、以下のコマンドを実行します。

```
# dnf install --enablerepo=codeready-builder-for-rhel-8-x86_64-rpms  python38-devel
```

検証手順

1. コンテナー内で有効なリポジトリーの一覧を表示します。

```
# dnf repolist
```

2. 必要なリポジトリーが一覧表示されていることを確認します。
3. インストール済みパッケージの一覧を表示します。

```
# rpm -qa
```

4. 必要なパッケージが一覧表示されていることを確認します。

注記

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、サブスクライブしている RHEL システム以外でコンテナーを配布する機能が限定される可能性があります。

11.3. 最小 UBI コンテナーへのソフトウェアの追加

UBI dnf 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。

手順

1. **registry.access.redhat.com/ubi9/ubi-minimal** イメージをプルして実行します。

```
$ podman run -it --name myubimin registry.access.redhat.com/ubi9/ubi-minimal
```

2. **myubimin** コンテナーにパッケージを追加します。

- UBI リポジトリーにあるパッケージを追加するには、リポジトリーを無効にしないでください。たとえば、**bzip2** パッケージを追加するには、次のコマンドを実行します。

```
# microdnf install bzip2
```

- 別のホストリポジトリーにあるパッケージを追加するには、必要なリポジトリーを明示的に有効にします。たとえば、**codeready-builder-for-rhel-8-x86_64-rpms** リポジトリーから **python38-devel** パッケージをインストールするには、以下のコマンドを実行します。

```
# microdnf install --enablerepo=codeready-builder-for-rhel-8-x86_64-rpms  python38-devel
```

検証手順

1. コンテナー内で有効なリポジトリーの一覧を表示します。

```
# microdnf repolist
```

2. 必要なリポジトリーが一覧表示されていることを確認します。
3. インストール済みパッケージの一覧を表示します。

```
# rpm -qa
```

4. 必要なパッケージが一覧表示されていることを確認します。

注記

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、サブスクライブしている RHEL システム以外でコンテナーを配布する機能が限定される可能性があります。

11.4. 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 システムにソフトウェアパッケージを追加するときに、リポジトリーを無効にする必要はありません。

手順

- UBI 標準または UBI init イメージに基づいて、実行中のコンテナーにパッケージを追加します。リポジトリーを無効にしないでください。**podman run** コマンドを使用してコンテナーを実行し、コンテナー内で **dnf install** コマンドを使用します。
  - たとえば、**bzip2** パッケージを UBI 標準ベースのコンテナーに追加するには、次のコマンドを実行します。

```
$ podman run -it --name myubi registry.access.redhat.com/ubi9/ubi
# dnf install bzip2
```

1. たとえば、**bzip2** パッケージを UBI init ベースのコンテナーに追加するには、次のコマンドを実行します。

```
$ podman run -it --name myubimin registry.access.redhat.com/ubi9/ubi-minimal
# microdnf install bzip2
```

検証手順

1. 有効なリポジトリーを一覧表示します。

   - UBI 標準イメージまたは UBI init イメージに基づいて、コンテナー内で有効なリポジトリーをすべて一覧表示するには、次のコマンドを実行します。

     ```
     #  dnf repolist
     ```

   - UBI の最小コンテナーに基づいて、コンテナー内で有効なリポジトリーを一覧表示するには、以下を実行します。

     ```
     # microdnf repolist
     ```

2. 必要なリポジトリーが一覧表示されていることを確認します。

3. インストール済みパッケージの一覧を表示します。

   ```
   # rpm -qa
   ```

4. 必要なパッケージが一覧表示されていることを確認します。

11.5. UBI ベースのイメージの構築

Buildah ユーティリティーを使用して、**Containerfile** から UBI ベースの Web サーバーコンテナーを作成できます。非 UBI dnf リポジトリーをすべて無効にして、イメージに再配布できる Red Hat ソフトウェアのみが含まれていることを確認する必要があります。

注記

UBI の最小イメージの場合は、**dnf** 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
```

手順

1. **Containerfile** を作成します。

   ```
   FROM registry.access.redhat.com/ubi9/ubi
   USER root
   LABEL maintainer="John Doe"
   # Update image
   RUN dnf update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos -
   y && rm -rf /var/cache/yum
   RUN dnf 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. コンテナーイメージをビルドします。

```
# buildah bud -t johndoe/webserver .
STEP 1: FROM registry.access.redhat.com/ubi9/ubi:latest
STEP 2: USER root
STEP 3: LABEL maintainer="John Doe"
STEP 4: RUN dnf update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos -y
...
Writing manifest to image destination
Storing signatures
--> f9874f27050
f9874f270500c255b950e751e53d37c6f8f6dba13425d42f30c2a8ef26b769f2
```

検証手順

1. Web サーバーを実行します。

```
# podman run -d --name=myweb -p 80:80 johndoe/webserver
bbe98c71d18720d966e4567949888dc4fb86eec7d304e785d5177168a5965f64
```

2. Web サーバーをテストします。

```
# curl http://localhost/index.html
The Web Server is Running
```

11.6. アプリケーションストリームランタイムイメージの使用

Application Streams に基づくランタイムイメージは、コンテナービルドのベースとして使用できる一連のコンテナーイメージを提供します。

サポートされるランタイムイメージは Python、Ruby、s2-core、s2i-base、.NET Core、PHP です。ランタイムイメージは Red Hat Container Catalog で利用できます。

注記

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。

関連情報

- Red Hat Container Catalog
- Red Hat Container Image Updates

11.7. 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イメージの形で入手できます。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** コマンドではなく、**skopeo** コマンドを使用します。

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 9 コンテナーでは、**registry.access.redhat.com/ubi9:8.1-397** 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/ubi9:8.1-397-source**)。

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/ubi9:8.1-397-source \
dir:$HOME/TEST
...
Copying blob 477bc8106765 done
Copying blob c438818481d3 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します。

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

4. 結果を確認します。

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

結果が正しい場合は、イメージを使用できる状態になります。

注記

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。

関連情報

- **skopeo-copy** の man ページ
- **skopeo-inspect** の man ページ

第12章 コンテナーでの SKOPEO、BUILDAH、および PODMAN の実行

本章では、コンテナーで Skopeo、Buildah、および Podman を実行する方法について説明します。

Skopeo を使用すると、すべてのレイヤーでイメージ全体をダウンロードすることなく、リモートレジストリーのイメージを検査できます。また、Sopeo を使用して、イメージのコピー、イメージの署名、イメージの同期、さまざまな形式およびレイヤー圧縮でのイメージ変換を行うこともできます。

Buildah を使用すると、OCI コンテナーイメージのビルドが容易になります。Buildah を使用すると、作業コンテナーをゼロから作成することも、イメージを開始点として使用して作成することも可能です。作業コンテナーからか、**Containerfile** の指示を使用して、イメージを作成できます。作業コンテナーの root ファイルシステムをマウントおよびアンマウントできます。

Podman を使用すると、コンテナーおよびイメージ、それらのコンテナーにマウントされたボリューム、およびコンテナーのグループから作成された Pod を管理できます。Podman は、コンテナーライフサイクル管理の **libpod** ライブラリーに基づいています。**libpod** ライブラリーは、コンテナー、Pod、コンテナーイメージ、およびボリュームを管理するための API を提供します。

コンテナーで Buildah、Skopeo、および Podman を実行する理由:

- **CI/CD システム**:
  - **Podman および Skopeo**: Kubernetes で CI/CD システムを実行するか、OpenShift を使用してコンテナーイメージをビルドし、これらのイメージを異なるコンテナーレジストリーに配布できます。Skopeo を Kubernetes ワークフローに統合するには、Skopeo をコンテナーで実行する必要があります。
  - **Buildah**:常にイメージをビルドする Kubernetes または OpenShift CI/CD システムに OCI/コンテナーイメージをビルドします。以前は、Docker ソケットを使用してコンテナーエンジンに接続し、**docker build** コマンドを実行していました。これは、パスワードなしにシステムに root アクセスを提供するのと同じで、安全ではありません。このため、Red Hat はコンテナーで Buildah を使用することを推奨します。
- **各種バージョン**:
  - **All**: ホストで古い OS を実行していても、最新バージョンの Skopeo、Buildah、または Podman を実行します。このソリューションは、コンテナーでコンテナーツールを実行します。たとえば、これは、最新版をネイティブで使用できない RHEL 7 コンテナーホストで、RHEL 8 で提供される最新版のコンテナーツールを実行するのに役立ちます。
- **HPC 環境**:
  - **All**: HPC 環境で一般的な制限は、root 以外のユーザーがホストにパッケージをインストールできないことです。コンテナーで Skopeo、Buildah、または Podman を実行すると、root 以外のユーザーとしてこの特定のタスクを実行することができます。

12.1. コンテナーでの SKOPEO の実行

この手順では、Skopeo を使用してリモートコンテナーイメージを検証する方法を説明します。コンテナーで Skopeo を実行すると、コンテナーの root ファイルシステムがホストの root ファイルシステムから分離されることを意味します。ホストとコンテナー間でファイルを共有またはコピーするには、ファイルとディレクトリーをマウントする必要があります。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

1. registry.redhat.io レジストリーにログインします。

```
$ podman login registry.redhat.io
Username: myuser@mycompany.com
Password: ***********
Login Succeeded!
```

2. **registry.redhat.io/rhel9/skopeo** コンテナーイメージを取得します。

```
$ podman pull registry.redhat.io/rhel9/skopeo
```

3. Skopeo を使用して、リモートコンテナーイメージ **registry.access.redhat.com/ubi9/ubi** を検査します。

```
$ podman run --rm registry.redhat.io/rhel9/skopeo skopeo inspect
docker://registry.access.redhat.com/ubi9/ubi
{
    "Name": "registry.access.redhat.com/ubi9/ubi",
    ...
    "Labels": {
        "architecture": "x86_64",
        ...
        "name": "ubi9",
        ...
        "summary": "Provides the latest release of Red Hat Universal Base Image 9.",
        "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/ubi9/images/8.2-347",
        ...
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
    ...
    ],
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "container=oci"
    ]
}
```

--rm オプションは、コンテナーの終了後に **registry.redhat.io/rhel9/skopeo** イメージを削除します。

関連情報

- How to run skopeo in a container

12.2. 認証情報を使用したコンテナーでの SKOPEO の実行

コンテナーレジストリーを使用する場合には、データへのアクセスや変更に認証が必要です。skopeo は、さまざまな認証情報の指定方法に対応します。

このアプローチでは、**--cred USERNAME[:PASSWORD]** オプションを使用してコマンドラインで認証情報を指定できます。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

- ロックされたレジストリーに対して Skopeo を使用してリモートコンテナーイメージを検証します。

```
$ podman run --rm registry.redhat.io/rhel9/skopeo inspect --creds $USER:$PASSWORD
docker://$IMAGE
```

関連情報

- How to run skopeo in a container

12.3. AUTHFILE を使用したコンテナーでの SKOPEO の実行

認証ファイル (authfile) を使用して認証情報を指定できます。**skopeo login** コマンドは、特定のレジストリーにログインし、認証トークンを authfile に保存します。authfiles を使用する利点は、認証情報を繰り返し入力する必要がないことです。

同じホストで実行する場合、Sopeo、Buildah、Podman などのすべてのコンテナーツールで同じ authfile を共有します。コンテナーで Skopeo を実行する場合は、コンテナーに authfile をボリュームマウントしてホスト上で authfile を共有するか、コンテナー内で再認証する必要があります。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

- ロックされたレジストリーに対して Skopeo を使用してリモートコンテナーイメージを検証します。

```
$ podman run --rm -v $AUTHFILE:/auth.json registry.redhat.io/rhel9/skopeo inspect
docker://$IMAGE
```

-v $AUTHFILE:/auth.json オプションの volume-mounts は、コンテナー内の /auth.json に authfile をマウントします。Skopeo は、ホストの authfile の認証トークンにアクセスでき、レジストリーにセキュアにアクセスできるようになります。

その他の Skopeo コマンドは、以下の例のように機能します。

- **skopeo-copy** コマンドで **--source-creds** および **--dest-creds** オプションを使用して、ソースおよび宛先のイメージに、コマンドラインで認証情報を指定します。また、**/auth.json** の authfile も読み取ります。
- ソースイメージおよび宛先イメージに個別の authfile を指定する場合は、**--source-authfile** オプションおよび **--dest-authfile** オプションを使用し、ホストからコンテナーにこれらの authfile をボリュームマウントします。

関連情報

- How to run skopeo in a container

12.4. ホストに対するコンテナーイメージのコピー

skopeo、Buildah、Podman は同じローカルコンテナーイメージストレージを共有します。ホストコンテナーストレージとの間でコンテナーをコピーする場合は、これを Skopeo コンテナーにマウントする必要があります。

注記

ホストコンテナーストレージへのパスは、root (**/var/lib/containers/storage**) と root 以外のユーザー (**$HOME/.local/share/containers/storage**) の間で異なります。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

1. **registry.access.redhat.com/ubi9/ubi** イメージをローカルストレージにコピーします。

```
$ podman run --privileged --rm -v
$HOME/.local/share/containers/storage:/var/lib/containers/storage
registry.redhat.io/rhel9/skopeo skopeo copy docker://registry.access.redhat.com/ubi9/ubi
containers-storage:registry.access.redhat.com/ubi9/ubi
```

   - **--privileged** オプションは、すべてのセキュリティーメカニズムを無効にします。Red Hat では、このオプションは信頼できる環境でのみ使用することを推奨します。
   - セキュリティーメカニズムを無効にするには、イメージを tarball またはその他のパスベースのイメージトランスポートにエクスポートして Skopeo コンテナーにマウントします。
     - **$ podman save --format oci-archive -o oci.tar $IMAGE**
     - **$ podman run --rm -v oci.tar:/oci.tar registry.redhat.io/rhel9/skopeo copy oci-archive:/oci.tar $DESTINATION**
2. オプション:ローカルストレージ内のイメージを一覧表示する

```
$ podman images
REPOSITORY                              TAG     IMAGE ID      CREATED       SIZE
registry.access.redhat.com/ubi9/ubi     latest  ecbc6f53bba0  8 weeks ago   211 MB
```

関連情報

- How to run skopeo in a container

12.5. コンテナーでの BUILDAH の実行

この手順では、コンテナーで Buildah を実行し、イメージを基に作業コンテナーを作成する方法を説明します。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

1. registry.redhat.io レジストリーにログインします。

```
$ podman login registry.redhat.io
Username: myuser@mycompany.com
Password: ***********
Login Succeeded!
```

2. **registry.redhat.io/rhel9/buildah** イメージをプルして実行します。

```
# podman run --rm --device /dev/fuse -it registry.redhat.io/rhel9/buildah  /bin/bash
```

   - **--rm** オプションは、コンテナーの終了後に **registry.redhat.io/rhel9/buildah** イメージを削除します。
   - **--device** オプションは、ホストデバイスをコンテナーに追加します。

3. **registry.access.redhat.com/ubi9** イメージを使用して、新しいコンテナーを作成します。

```
# buildah from registry.access.redhat.com/ubi9
...
ubi9-working-container
```

4. **ubi9-working-container** コンテナー内で **ls** / コマンドを実行します。

```
# buildah run  --isolation=chroot ubi9-working-container ls /
bin  boot  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv
```

5. オプション:ローカルストレージ内のイメージを一覧表示します。

```
# buildah images
REPOSITORY                      TAG      IMAGE ID       CREATED       SIZE
registry.access.redhat.com/ubi9   latest   ecbc6f53bba0   5 weeks ago   211 MB
```

6. オプション:作業用のコンテナーおよびそれらのベースイメージを一覧表示します。

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
0aaba7192762     *     ecbc6f53bba0 registry.access.redhat.com/ub... ubi9-working-container
```

7. オプション:**registry.access.redhat.com/ubi9** イメージを **registry.example.com** にあるローカルレジストリーにプッシュします。

```
# buildah push ecbc6f53bba0 registry.example.com:5000/ubi9/ubi
```

関連情報

- Best practices for running Buildah in a container

12.6. 特権および非特権 PODMAN コンテナー

デフォルトでは、Podman コンテナーは権限がないため、たとえばホスト上のオペレーティングシステムの一部を変更することはできません。これは、デフォルトでは、コンテナーはデバイスへの限定的なアクセスしか許可されないためです。

以下は、特権コンテナーの重要なプロパティーの一覧です。特権コンテナーは、**podman run --privileged <image_name>** コマンドを使用して実行できます。

- 特権コンテナーには、コンテナーを起動するユーザーと同じデバイスへのアクセス権限が付与されます。
- 特権コンテナーは、コンテナーをホストから分離するセキュリティー機能を無効にします。削除された機能、制限されたデバイス、読み取り専用マウントポイント、Apparmor/SELinux の分離、および Seccomp フィルターは、すべて無効にされます。
- 特権コンテナーには、それらを起動したアカウント以上の特権を割り当てることはできません。

関連情報

- How to use the --privileged flag with container engines
- **podman-run** の man ページ

12.7. 拡張された権限での PODMAN の実行

ルートレス環境でワークロードを実行できない場合は、これらのワークロードを root ユーザーとして実行する必要があります。拡張された権限でのコンテナーの実行は、慎重に実施する必要があります。すべてのセキュリティー機能が無効になるためです。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

- Podman コンテナー内で Podman コンテナーを実行します。

```
$ podman run --privileged --name=privileged_podman registry.access.redhat.com//podman
podman run ubi9 echo hello
Resolved "ubi9" as an alias (/etc/containers/registries.conf.d/001-rhel-shortnames.conf)
Trying to pull registry.access.redhat.com/ubi9:latest...
...
Storing signatures
hello
```

- **registry.access.redhat.com/rhel9/podman** イメージに基づいて、**privileged_podman** という名前の外部コンテナーを実行します。
- **--privileged** オプションは、コンテナーをホストから分離するセキュリティー機能を無効にします。
- **podman run ubi9 echo hello** コマンドを実行して、**ubi9** イメージに基づいて内部コンテナーを作成します。
- **ubi9** の短縮イメージ名がエイリアスとして解決されていることに注意してください。これにより、**registry.access.redhat.com/ubi9:latest** イメージがプルされます。

検証

- すべてのコンテナーを一覧表示します。

```
$ podman ps -a
CONTAINER ID  IMAGE                                COMMAND               CREATED         STATUS
PORTS   NAMES
52537876caf4  registry.access.redhat.com/ubi9/podman  podman run ubi9 e...  30
seconds ago     Exited (0) 13 seconds ago               privileged_podman
```

関連情報

- How to use Podman inside of a container
- **podman-run** の man ページ

12.8. 少ない権限での PODMAN の実行

--privileged オプションを指定せずに、ネストされた 2 つの Podman コンテナーを実行できます。**--privileged** オプションを指定せずにコンテナーを実行することは、より安全なオプションです。

これは、可能な限り安全な方法で異なるバージョンの Podman を試す場合に役立ちます。

前提条件

- **container-tools** モジュールがインストールされている。

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

手順

- ネストされた 2 つのコンテナーを実行します。

```
$ podman run --name=unprivileged_podman --security-opt label=disable --user podman --
device /dev/fuse registry.access.redhat.com/ubi9/podman podman run ubi9 echo hello
```

- **registry.access.redhat.com/ubi9/podman** イメージに基づいて、**unprivileged_podman** という名前の外部コンテナーを実行します。
- **--security-opt label=disable** オプションは、ホスト Podman での SELinux の分離を無効にします。SELinux では、コンテナー化されたプロセスは、コンテナー内で実行するために必要なすべてのファイルシステムをマウントすることはできません。
- **--user podman** オプションを指定すると、自動的に、外部コンテナー内の Podman がユーザーの名前空間内で実行されるようになります。
- **--device /dev/fuse** オプションは、コンテナー内の **fuse-overlayfs** パッケージを使用します。このオプションは **/dev/fuse** を外部コンテナーに追加し、コンテナー内の Podman がこれを使用できるようにします。
- **podman run ubi9 echo hello** コマンドを実行して、**ubi9** イメージに基づいて内部コンテナーを作成します。
- ubi9 の短縮イメージ名がエイリアスとして解決されていることに注意してください。これにより、**registry.access.redhat.com/ubi9:latest** イメージがプルされます。

検証

- すべてのコンテナーを一覧表示します。

```
$ podman ps -a
CONTAINER ID  IMAGE                         COMMAND           CREATED          STATUS
PORTS   NAMES
a47b26290f43              podman run ubi9 e...  30 seconds ago     Exited (0) 13 seconds ago
unprivileged_podman
```

12.9. PODMAN コンテナー内でのコンテナーのビルド

この手順では、Podman を使用してコンテナー内でコンテナーを実行する方法を説明します。この例では、Podman を使用してコンテナーから別のコンテナーをビルドし、実行する方法を示しています。コンテナーは、簡単なテキストベースのゲーム「Moon-buggy」を実行します。

前提条件

- **container-tools** モジュールがインストールされている。

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

- registry.redhat.io レジストリーにログインしている。

```
# podman login registry.redhat.io
```

手順

1. **registry.redhat.io/rhel9/podman** イメージをベースとするコンテナーを実行します。

```
# podman run --privileged --name podman_container -it registry.redhat.io/rhel9/podman
/bin/bash
```

   - **registry.access.redhat.com/rhel9/podman** イメージに基づいて、**privileged_podman** という名前の外部コンテナーを実行します。
   - **--it** オプションは、コンテナー内で対話式 bash シェルを実行します。
   - **--privileged** オプションは、コンテナーをホストから分離するセキュリティー機能を無効にします。

2. **podman_container** コンテナー内に **Containerfile** を作成します。

```
# vi Containerfile
FROM registry.access.redhat.com/ubi9/ubi
RUN dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
RUN dnf -y install moon-buggy && dnf clean all
CMD ["/usr/bin/moon-buggy"]
```

   Containerfile のコマンドで以下の build コマンドが実行されます。

   - **registry.access.redhat.com/ubi9/ubi** イメージからコンテナーをビルドします。
   - **epel-release-latest-8.noarch.rpm** パッケージをインストールします。
   - **moon-buggy** パッケージをインストールします。
   - コンテナーコマンドを設定します。

3. **Containerfile** を使用して **moon-buggy** という名前の新しいコンテナーイメージをビルドします。

```
# podman build -t moon-buggy .
```

4. オプション:すべてのイメージを一覧表示します。

```
# podman images
REPOSITORY                TAG      IMAGE ID      CREATED        SIZE
localhost/moon-buggy  latest  c97c58abb564  13 seconds ago  1.67 GB
registry.access.redhat.com/ubi9/ubi latest 4199acc83c6a  132seconds ago 213 MB
```

5. **moon-buggy** コンテナーに基づいて新しいコンテナーを実行します。

```
# podman run -it --name moon moon-buggy
```

6. オプション:**moon-buggy** イメージにタグを付けます。

```
# podman tag moon-buggy registry.example.com/moon-buggy
```

7. オプション:**moon-buggy** イメージをレジストリーにプッシュします。

```
# podman push registry.example.com/moon-buggy
```

関連情報

- テクノロジープレビュー: コンテナー内でコンテナーの実行

第13章 BUILDAH でコンテナーイメージの構築

Buildah は、OCI Runtime Specification を満たす OCI コンテナーイメージの構築を容易にします。Buildah を使用すると、作業コンテナーをゼロから作成することも、イメージを開始点として使用して作成することも可能です。作業コンテナーからか、**Containerfile** の指示を使用して、イメージを作成できます。作業コンテナーの root ファイルシステムをマウントおよびアンマウントできます。

13.1. BUILDAH ツール

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。

デーモンなし

Buildah にはコンテナーランタイムは必要ありません。

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ビルドしたり、空のイメージで始めることができます。

ビルドツールが外部のもの

Buildah では、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。

- ビルドイメージのサイズを縮小します。
- 作成されたイメージからソフトウェア（gcc、make、dnf など）を除外して、イメージのセキュリティーを強化します。
- イメージサイズの縮小により、少ないリソースを使用してイメージを転送できます。

互換性

Buildah は、Dockerfile でコンテナーイメージを構築することで、Docker から Buildah への移行が容易になります。

注記

Buildahがコンテナーストレージにデフォルトで使用する場所は、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ために使用する場所と同じです。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、CRI-O および Buildah が現在イメージを共有できても、コンテナーを共有することはできません。

関連情報

- Buildah - a tool that facilitates building Open Container Initiative (OCI) container images
- Buildah Tutorial 1:Building OCI container images
- Buildah Tutorial 2:Using Buildah with container registries
- Buildah を使用した構築: Dockerfile、コマンドライン、またはスクリプト
- ルートレス Buildah の仕組み: 非特権環境でのコンテナーの構築

13.2. BUILDAH のインストール

dnf コマンドを使用して Buildah ツールをインストールします。

手順

- Buildah ツールをインストールします。

```
# dnf -y install buildah
```

検証

- ヘルプメッセージを表示します。

```
# buildah -h
```

13.3. BUILDAH でイメージの取得

buildah from コマンドを使用して、新規作業コンテナーをゼロから作成するか、または指定したイメージを開始点として作成します。

手順

- **registry.redhat.io/ubi9/ubi** イメージをもとに、新しい作業コンテナーを作成します。

```
# buildah from registry.access.redhat.com/ubi9/ubi
Getting image source signatures
Copying blob…
Writing manifest to image destination
Storing signatures
ubi-working-container
```

検証

1. ローカルストレージ内のイメージを一覧表示します。

```
# buildah images
REPOSITORY                              TAG      IMAGE ID       CREATED       SIZE
registry.access.redhat.com/ubi9/ubi     latest   272209ff0ae5   2 weeks ago   234 MB
```

2. 作業用のコンテナーおよびそれらのベースイメージを一覧表示します。

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
01eab9588ae1     *     272209ff0ae5 registry.access.redhat.com/ub... ubi-working-container
```

関連情報

- **buildah-from** の man ページ
- **buildah-images** の man ページ
- **buildah-containers** の man ページ

13.4. コンテナー内でのコマンドの実行

buildah run コマンドを使用して、コンテナーからコマンドを実行します。

前提条件

- プルしたイメージがローカルシステムで利用できる。

手順

- オペレーティングシステムのバージョンを表示します。

```
# buildah run ubi-working-container cat /etc/redhat-release
Red Hat Enterprise Linux release 8.4 (Ootpa)
```

関連情報

- **buildah-run** の man ページ

13.5. BUILDAH を使用した CONTAINERFILE からのイメージのビルド

buildah bud コマンドを使用して、**Containerfile** からのステップを使用してイメージをビルドします。

注記

buildah bud コマンドは、コンテキストディレクトリーにある場合に、**Containerfile** を使用します。コンテキストディレクトリーにない場合、**buildah bud** コマンドは **Dockerfile** を使用します。それ以外の場合は、**--file** オプションでファイルを指定できます。**Containerfile** および **Dockerfile** 内で使用できる利用可能なコマンドは同じです。

手順

1. **Containerfile** を作成します。

```
# cat Containerfile
FROM registry.access.redhat.com/ubi9/ubi
ADD myecho /usr/local/bin
ENTRYPOINT "/usr/local/bin/myecho"
```

2. **myecho** スクリプトを作成します。

```
# cat myecho
echo "This container works!"
```

3. **myecho** スクリプトのアクセスパーミッションを変更します。

```
# chmod 755 myecho
```

4. 現在のディレクトリーの **Containerfile** を使用して **myecho** イメージをビルドします。

```
# buildah bud -t myecho .
STEP 1: FROM registry.access.redhat.com/ubi9/ubi
```

```
STEP 2: ADD myecho /usr/local/bin
STEP 3: ENTRYPOINT "/usr/local/bin/myecho"
STEP 4: COMMIT myecho
...
Storing signatures
```

検証

1. すべてのイメージを一覧表示します。

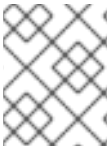
```
# buildah images
REPOSITORY                TAG      IMAGE ID       CREATED              SIZE
localhost/myecho          latest   b28cd00741b3   About a minute ago   234 MB
```

2. **localhost/myecho** イメージに基づいて **myecho** コンテナーを実行します。

```
# podman run --name=myecho localhost/myecho
This container works!
```

3. すべてのコンテナーを一覧表示します。

```
# podman ps -a
0d97517428d  localhost/myecho                        12 seconds ago  Exited (0) 13
seconds ago          myecho
```



注記

podman history コマンドを使用して、イメージで使用された各レイヤーに関する情報を表示できます。

関連情報

- **buildah-bud** の man ページ

13.6. BUILDAH でコンテナーおよびイメージの検証

buildah inspect コマンドを使用して、コンテナーまたはイメージに関する情報を表示します。

前提条件

- イメージが Containerfile の命令を使用してビルドされている。詳細は、「Buildah を使用した Containerfile からのイメージのビルド」を参照してください。

手順

- イメージを検証します。
  - myecho イメージを検証するには、以下を入力します。

```
# buildah inspect localhost/myecho
 {
    "Type": "buildah 0.0.1",
    "FromImage": "localhost/myecho:latest",
```

```
    "FromImageID":
"b28cd00741b38c92382ee806e1653eae0a56402bcd2c8d31bdcd36521bc267a4",
    "FromImageDigest":
"sha256:0f5b06cbd51b464fabe93ce4fe852a9038cdd7c7b7661cd7efef8f9ae8a59585",
    "Config":
    ...
     "Entrypoint": [
                "/bin/sh",
                "-c",
                "\"/usr/local/bin/myecho\""
            ],
    ...
}
```

- **myecho** イメージから作業コンテナーを検証するには、以下を実行します。
  i. **localhost/myecho** イメージをもとに作業コンテナーを作成します。

```
# buildah from localhost/myecho
```

  ii. **myecho-working-container** コンテナーを検査します。

```
# buildah inspect ubi-working-container
{
    "Type": "buildah 0.0.1",
    "FromImage": "registry.access.redhat.com/ubi8/ubi:latest",
    "FromImageID":
"272209ff0ae5fe54c119b9c32a25887e13625c9035a1599feba654aa7638262d",
    "FromImageDigest":
"sha256:77623387101abefbf83161c7d5a0378379d0424b2244009282acb39d42f1fe13
",
    "Config":
    ...
"Container": "ubi-working-container",
"ContainerID":
"01eab9588ae1523746bb706479063ba103f6281ebaeeccb5dc42b70e450d5ad0",
"ProcessLabel": "system_u:system_r:container_t:s0:c162,c1000",
"MountLabel": "system_u:object_r:container_file_t:s0:c162,c1000",
...
}
```

関連情報

- **buildah-inspect** の man ページ

13.7. BUILDAH MOUNT を使用したコンテナーの変更

buildah inspect コマンドを使用して、コンテナーまたはイメージに関する情報を表示します。

前提条件

- Containerfile の指示を使用してビルドされたイメージ。詳細は、「Buildah を使用した Containerfile からのイメージのビルド」を参照してください。

手順

1. **registry.access.redhat.com/ubi8/ubi** イメージをもとに作業コンテナーを作成し、コンテナーの名前を **mycontainer** 変数に保存します。

```
# mycontainer=$(buildah from localhost/myecho)

# echo $mycontainer
myecho-working-container
```

2. **myecho-working-container** コンテナーをマウントし、マウントポイントパスを **mymount** 変数に保存します。

```
# mymount=$(buildah mount $mycontainer)

# echo $mymount
/var/lib/containers/storage/overlay/c1709df40031dda7c49e93575d9c8eebcaa5d8129033a58e5
b6a95019684cc25/merged
```

3. **myecho** スクリプトを変更し、実行可能にします。

```
# echo 'echo "We modified this container."' >> $mymount/usr/local/bin/myecho
# chmod +x $mymount/usr/local/bin/myecho
```

4. **myecho-working-container** コンテナーから **myecho2** イメージを作成します。

```
# buildah commit $mycontainer containers-storage:myecho2
```

検証

1. ローカルストレージ内のイメージを一覧表示します。

```
# buildah images
REPOSITORY                      TAG      IMAGE ID       CREATED          SIZE
docker.io/library/myecho2         latest   4547d2c3e436   4 minutes ago    234 MB
localhost/myecho                latest   b28cd00741b3   56 minutes ago   234 MB
```

2. **docker.io/library/myecho2** イメージに基づいて **myecho2** コンテナーを実行します。

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

関連情報

- **buildah-mount** の man ページ
- **buildah-commit** の man ページ

13.8. BUILDAH COPY および BUILDAH CONFIG を使用したコンテナーの変更

buildah copy コマンドを使用して、マウントせずにファイルをコンテナーにコピーします。次に、**buildah config** コマンドを使用して、デフォルトで作成したスクリプトを実行するコンテナーを設定できます。

前提条件

- Containerfile の指示を使用してビルドされたイメージ。詳細は、「Buildah を使用した Containerfile からのイメージのビルド」を参照してください。

手順

1. **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
```

2. 新しい作業コンテナーを作成します。

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

3. newecho スクリプトを、コンテナー内の **/usr/local/bin** ディレクトリーにコピーします。

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

4. **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
```

5. オプション。実行する**newecho** スクリプトのトリガーとなる**myecho-working-container-2** コンテナーを実行します。

```
# buildah run myecho-working-container-2 -- sh -c '/usr/local/bin/newecho'
I changed this container
```

6. **myecho-working-container-2** コンテナーを、**mynewecho** という新しいイメージにコミットします。

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

検証

- ローカルストレージ内のイメージを一覧表示します。

```
# buildah images
REPOSITORY                      TAG      IMAGE ID       CREATED         SIZE
docker.io/library/mynewecho     latest   fa2091a7d8b6   8 seconds ago   234 MB
```

関連情報

- **buildah-copy** の man ページ

- **buildah-config** の man ページ
- **buildah-commit** の man ページ
- **buildah-run** の man ページ

13.9. BUILDAH でゼロ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、最低限のコンテナーメタデータのみを保持する新しいコンテナーを作成できます。

スクラッチコンテナーからイメージを作成するときは、次のことを考慮してください。

- 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。
- RPM データベースを初期化し、**dnf** や **rpm** などのツールを使用するために、コンテナーにリリースパッケージを追加する必要があります。
- パッケージを多数追加する場合は、scratch イメージではなく、標準の UBI イメージまたは最小 UBI イメージの使用を検討してください。

手順

この手順では、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。

1. 空のコンテナーを作成します。

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

2. **working-container** コンテナーをマウントし、マウントポイントパスを **scratchmnt** 変数に保存します。

```
# scratchmnt=$(buildah mount working-container)


# echo $scratchmnt
/var/lib/containers/storage/overlay/be2eaecf9f74b6acfe4d0017dd5534fde06b2fa8de9ed875691f6ccc791c1836/merged
```

3. 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。

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

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

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

5. **$scratchmnt/var/www/html/index.html** ファイルを作成します。

```
# mkdir -p $scratchmnt/var/www/html
# echo "Your httpd container from scratch works!" > $scratchmnt/var/www/html/index.html
```

6. コンテナーから直接 **httpd** デーモンを実行するように **working-container** 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
```

検証

1. ローカルストレージ内のイメージを一覧表示します。

```
# podman images
REPOSITORY                  TAG     IMAGE ID      CREATED         SIZE
localhost/myhttpd           latest  08da72792f60  2 minutes ago   121 MB
```

2. **localhost/myhttpd** イメージを実行し、コンテナーとホストシステム間のポートマッピングを設定します。

```
# podman run -p 8080:80 -d --name myhttpd 08da72792f60
```

3. Web サーバーをテストします。

```
# curl localhost:8080
Your httpd container from scratch works!
```

関連情報

- **buildah-config** の man ページ
- **buildah-commit** の man ページ

13.10. プライベートレジストリーへのコンテナーのプッシュ

buildah push コマンドを使用して、イメージをローカルストレージからパブリックリポジトリーまたはプライベートリポジトリーにプッシュします。

前提条件

- イメージが Containerfile の命令を使用してビルドされている。詳細は、「Buildah を使用した Containerfile からのイメージのビルド」を参照してください。

手順

1. マシンにローカルレジストリーを作成します。

```
# podman run -d -p 5000:5000 registry:2
```

2. **myecho:latest** イメージを **localhost** レジストリーにプッシュします。

```
#  buildah push --tls-verify=false myecho:latest localhost:5000/myecho:latest
Getting image source signatures
Copying blob sha256:e4efd0...
```

```
...
Writing manifest to image destination
Storing signatures
```

検証

1. **localhost** リポジトリー内のイメージを一覧表示します。

```
# curl http://localhost:5000/v2/_catalog
{"repositories":["myecho2]}


# curl http://localhost:5000/v2/myecho2/tags/list
{"name":"myecho","tags":["latest"]}
```

2. **docker://localhost:5000/myecho:latest** イメージを調査します。

```
# skopeo inspect --tls-verify=false docker://localhost:5000/myecho:latest | less
{
    "Name": "localhost:5000/myecho",
    "Digest": "sha256:8999ff6050...",
    "RepoTags": [
        "latest"
    ],
    "Created": "2021-06-28T14:44:05.919583964Z",
    "DockerVersion": "",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.redhat.io",
    ...
}
```

3. **localhost:5000/myecho** イメージをプルします。

```
# podman pull --tls-verify=false localhost:5000/myecho2
# podman run localhost:5000/myecho2
This container works!
```

関連情報

- **buildah-push** の man ページ

13.11. 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します。

前提条件

- Containerfile の指示を使用してビルドされたイメージ。詳細は、「Buildah を使用した Containerfile からのイメージのビルド」を参照してください。

手順

1. **docker.io/library/myecho:latest** を Docker Hub にプッシュします。 **username** および **password** を、Docker Hub 認証情報に置き換えます。

```
# buildah push --creds username:password \
        docker.io/library/myecho:latest docker://testaccountXX/myecho:latest
```

検証

- **docker.io/testaccountXX/myecho:latest** イメージを取得して実行します。
  - Podman ツールの使用:

```
# podman run docker.io/testaccountXX/myecho:latest
This container works!
```

  - Buildah および Podman ツールの使用:

```
# buildah from docker.io/testaccountXX/myecho:latest
myecho2-working-container-2
# podman run myecho-working-container-2
```

関連情報

- **buildah-push** の man ページ

13.12. BUILDAH でイメージの削除

buildah rmi コマンドを使用して、ローカルに保存されたコンテナーイメージを削除します。ID または名前を使用して、イメージを削除できます。

手順

1. ローカルシステムにある全イメージの一覧を表示します。

```
# buildah images
REPOSITORY                                TAG      IMAGE ID       CREATED          SIZE
localhost/johndoe/webserver               latest   dc5fcc610313   46 minutes ago   263 MB
docker.io/library/mynewecho               latest   fa2091a7d8b6   17 hours ago     234 MB
docker.io/library/myecho2                 latest   4547d2c3e436   6 days ago       234 MB
localhost/myecho                          latest   b28cd00741b3   6 days ago       234 MB
localhost/ubi-micro-httpd                 latest   c6a7678c4139   12 days ago      152 MB
registry.access.redhat.com/ubi9/ubi       latest   272209ff0ae5   3 weeks ago      234 MB
```

2. **localhost/myecho** イメージを削除します。

```
# buildah rmi localhost/myecho
```

- 複数のイメージを削除するには、以下のコマンドを実行します。

```
# buildah rmi docker.io/library/mynewecho  docker.io/library/myecho2
```

- システムからすべてのイメージを削除するには、以下のコマンドを実行します。

```
# buildah rmi -a
```

- 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、**-f** オプションを追加して削除します。

```
# buildah rmi -f localhost/ubi-micro-httpd
```

検証

- イメージが削除されていることを確認します。

```
# buildah images
```

関連情報

- **buildah-rmi** の man ページ

13.13. BUILDAH でコンテナーの削除

buildah rm コマンドを使用して、コンテナーを削除します。コンテナー ID または名前で、削除するコンテナーを指定できます。

前提条件

- 1 つ以上のコンテナーが停止されている。

手順

1. すべてのコンテナーを一覧表示します。

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
05387e29ab93     *     c37e14066ac7 docker.io/library/myecho:latest  myecho-working-
container
```

2. myecho-working-container コンテナーを削除します。

```
# buildah rm myecho-working-container
05387e29ab93151cf52e9c85c573f3e8ab64af1592b1ff9315db8a10a77d7c22
```

検証

- コンテナーが削除されていることを確認します。

```
# buildah containers
```

関連情報

- **buildah-rm** の man ページ

第14章 コンテナーの監視

本章では、コンテナーの正常性の判別、システムおよび Pod 情報の表示、Podman イベントの監視など、Podman 環境を管理できる便利な Podman コマンドについて主に説明します。

14.1. コンテナーでの HEALTHCHECK の実行

Healthcheck を使用すると、コンテナー内で実行されるプロセスの健全性または準備状態を判別できます。Healthcheck は、以下の基本的なコンポーネント 5 つで構成されます。

- Command
- Retries
- Interval
- Start-period
- Timeout

以下は、healthcheck コンポーネントの説明です。

Command
Podman は、ターゲットコンテナー内でコマンドを実行して終了コードを待ちます。

他の 4 つのコンポーネントは任意で healthcheck のスケジューリングに関係があります。

Retries
「正常でない」とマークするまでに、ヘルスチェックで連続して不合格とならなければならないか、その回数を定義します。Healthcheck に成功すると、再試行数がリセットされます。

Interval
次回に healthcheck コマンドを実行するまでの時間を指定します。間隔が短いと、システムでの healthcheck の実行時間が長くなる点に注意してください。間隔が長いと、タイムアウトの検出が困難になります。

Start-period
コンテナーを起動してから healthcheck の不合格を無視するまでの時間を指定します。

Timeout
healthcheck の完了前に不合格とみなされるまでの期間を指定します。

Healthcheck はコンテナー内で実行されます。Healthcheck は、サービスが正常な状態を理解しており、ヘルスチェックの成功と不合格を区別できる場合にだけ役に立ちます。

手順

1. Healthcheck を定義します。

```
$ podman run -dt --name hc1 -p 8080:8080  --health-cmd='curl http://localhost:8080 || exit 1'
--health-interval=0 registry.access.redhat.com/ubi8/httpd-24
```

- **--health-cmd** オプションは、コンテナーの healthcheck コマンドを設定します。
- healthcheck を手動で実行するには、**-health-interval=0** オプション で 0 の値を指定します。

2. Healthcheck を手動で実行します。

```
$ podman healthcheck run hc1
Healthy
```

3. 必要に応じて、最後のコマンドの終了ステータスを確認できます。

```
$ echo $?
0
```

「0」値は成功したことを指します。

関連情報

- **podman-run** の man ページ
- Monitoring container vitality and availability with Podman

14.2. PODMAN システム情報の表示

podman system コマンドを使用すると、Podman システムを管理できます。このセクションでは、Podman システム情報を表示する方法を説明します。

手順

- Podman システム情報を表示します。
  - Podman のディスク使用量を表示するには、以下を入力します。

```
$ podman system df
TYPE           TOTAL       ACTIVE      SIZE        RECLAIMABLE
Images         3           2           1.085GB     233.4MB (0%)
Containers     2           0           28.17kB     28.17kB (100%)
Local Volumes  3           0           0B          0B (0%)
```

  - 領域の使用量に関する詳細情報を表示するには、次のコマンドを実行します。

```
$ podman system df -v
Images space usage:

REPOSITORY                            TAG         IMAGE ID      CREATED     SIZE
SHARED SIZE  UNIQUE SIZE  CONTAINERS
registry.access.redhat.com/ubi9       latest      b1e63aaae5cf  13 days     233.4MB
233.4MB      0B           0
registry.access.redhat.com/ubi9/httpd-24  latest      0d04740850e8  13 days     461.5MB
0B           461.5MB      1
registry.redhat.io/rhel8/podman       latest      dce10f591a2d  13 days     390.6MB
233.4MB      157.2MB      1

Containers space usage:

CONTAINER ID  IMAGE         COMMAND               LOCAL VOLUMES  SIZE
CREATED     STATUS      NAMES
311180ab99fb  0d04740850e8  /usr/bin/run-httpd    0              28.17kB     16 hours
```

```
exited      hc1
bedb6c287ed6  dce10f591a2d  podman run ubi9 echo hello  0             0B          11 hours
configured  dazzling_tu

Local Volumes space usage:

VOLUME NAME                                                 LINKS       SIZE
76de0efa83a3dae1a388b9e9e67161d28187e093955df185ea228ad0b3e435d0  0
0B
8a1b4658aecc9ff38711a2c7f2da6de192c5b1e753bb7e3b25e9bf3bb7da8b13  0
0B
d9cab4f6ccbcf2ac3cd750d2efff9d2b0f29411d430a119210dd242e8be20e26  0           0B
```

- Podman のホスト、現在のストレージ統計、およびビルドについての情報を表示するには、以下を入力します。

```
$ podman system info
host:
  arch: amd64
  buildahVersion: 1.22.3
  cgroupControllers: []
  cgroupManager: cgroupfs
  cgroupVersion: v1
  conmon:
    package: conmon-2.0.29-1.module+el8.5.0+12381+e822eb26.x86_64
    path: /usr/bin/conmon
    version: 'conmon version 2.0.29, commit:
7d0fa63455025991c2fc641da85922fde889c91b'
  cpus: 2
  distribution:
    distribution: '"rhel"'
    version: "8.5"
  eventLogger: file
  hostname: localhost.localdomain
  idMappings:
    gidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
    uidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
  kernel: 4.18.0-323.el8.x86_64
  linkmode: dynamic
  memFree: 352288768
  memTotal: 2819129344
  ociRuntime:
    name: runc
    package: runc-1.0.2-1.module+el8.5.0+12381+e822eb26.x86_64
```

```
    path: /usr/bin/runc
    version: |-
      runc version 1.0.2
      spec: 1.0.2-dev
      go: go1.16.7
      libseccomp: 2.5.1
  os: linux
  remoteSocket:
    path: /run/user/1000/podman/podman.sock
  security:
    apparmorEnabled: false
    capabilities:
CAP_NET_RAW,CAP_CHOWN,CAP_DAC_OVERRIDE,CAP_FOWNER,CAP_FSETID,C
AP_KILL,CAP_NET_BIND_SERVICE,CAP_SETFCAP,CAP_SETGID,CAP_SETPCAP,CA
P_SETUID,CAP_SYS_CHROOT
    rootless: true
    seccompEnabled: true
    seccompProfilePath: /usr/share/containers/seccomp.json
    selinuxEnabled: true
  serviceIsRemote: false
  slirp4netns:
    executable: /usr/bin/slirp4netns
    package: slirp4netns-1.1.8-1.module+el8.5.0+12381+e822eb26.x86_64
    version: |-
      slirp4netns version 1.1.8
      commit: d361001f495417b880f20329121e3aa431a8f90f
      libslirp: 4.4.0
      SLIRP_CONFIG_VERSION_MAX: 3
      libseccomp: 2.5.1
  swapFree: 3113668608
  swapTotal: 3124752384
  uptime: 11h 24m 12.52s (Approximately 0.46 days)
registries:
  search:
  - registry.fedoraproject.org
  - registry.access.redhat.com
  - registry.centos.org
  - docker.io
store:
  configFile: /home/user/.config/containers/storage.conf
  containerStore:
    number: 2
    paused: 0
    running: 0
    stopped: 2
  graphDriverName: overlay
  graphOptions:
    overlay.mount_program:
      Executable: /usr/bin/fuse-overlayfs
      Package: fuse-overlayfs-1.7.1-1.module+el8.5.0+12381+e822eb26.x86_64
      Version: |-
        fusermount3 version: 3.2.1
        fuse-overlayfs: version 1.7.1
        FUSE library version 3.2.1
        using FUSE kernel interface version 7.26
  graphRoot: /home/user/.local/share/containers/storage
```

```
  graphStatus:
    Backing Filesystem: xfs
    Native Overlay Diff: "false"
    Supports d_type: "true"
    Using metacopy: "false"
  imageStore:
    number: 3
  runRoot: /run/user/1000/containers
  volumePath: /home/user/.local/share/containers/storage/volumes
version:
  APIVersion: 3.3.1
  Built: 1630360721
  BuiltTime: Mon Aug 30 23:58:41 2021
  GitCommit: ""
  GoVersion: go1.16.7
  OsArch: linux/amd64
  Version: 3.3.1
```

- 未使用のコンテナー、イメージ、およびボリュームデータをすべて削除するには、次のコマンドを実行します。

```
$ podman system prune
WARNING! This will remove:
        - all stopped containers
        - all stopped pods
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
```

  - **podman system prune** コマンドは、未使用のコンテナー (関連付けられていないコンテナーおよび参照されていないコンテナー両方)、Pod、ローカルストレージのボリューム (任意) をすべて削除します。
  - **--all** オプションを使用して、未使用のイメージをすべて削除します。未使用のイメージとは、ベースとするコンテナーのないイメージや、関連付けられていないイメージを指します。
  - ボリュームをプルーニングするには、**--volume** オプションを使用します。デフォルトでは、現在ボリュームを使用するコンテナーがない場合に、重要なデータが削除されないようにボリュームは削除されません。

関連情報

- **podman-system-df** の man ページ
- **podman-system-info** の man ページ
- **podman-system-prune** の man ページ

14.3. PODMAN イベントタイプ

Podman で発生するイベントをモニターできます。イベントタイプが複数存在し、イベントタイプごとに異なるステータスを報告します。

コンテナー イベントタイプは以下のステータスを報告します。

- attach
- checkpoint
- cleanup
- commit
- create
- exec
- export
- import
- init
- kill
- mount
- pause
- prune
- remove
- restart
- restore
- start
- stop
- sync
- unmount
- unpause

Pod イベントタイプは以下のステータスを報告します。

- create
- kill
- pause
- remove
- start
- stop
- unpause

image イベントタイプは以下のステータスを報告します。

- prune
- push
- pull
- save
- remove
- tag
- untag

system タイプは、以下のステータスを報告します。

- refresh
- renumber

volume タイプは、以下のステータスを報告します。

- create
- prune
- remove

関連情報

- **podman-events** の man ページ

14.4. PODMAN イベントのモニタリング

Podman で発生するイベントを監視して出力できます。各イベントには、タイムスタンプ、タイプ、ステータス、名前 (該当する場合)、およびイメージ (該当する場合) が含まれます。

手順

- Podman イベントを表示します。
  - すべての Podman イベントを表示するには、以下を入力します。

```
$ podman events
2020-05-14 10:33:42.312377447 -0600 CST container create 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:46.958768077 -0600 CST container init 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:46.973661968 -0600 CST container start 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:50.833761479 -0600 CST container stop 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:51.047104966 -0600 CST container cleanup 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
```

ログを終了するには、CTRL+c を押します。

- Podman の作成イベントのみを表示するには、以下を入力します。

```
$ podman events --filter event=create
2020-05-14 10:36:01.375685062 -0600 CST container create 20dc581f6fbf
(image=registry.access.redhat.com/ubi8/ubi:latest)
2019-03-02 10:36:08.561188337 -0600 CST container create 58e7e002344c
(image=registry.access.redhat.com/ubi8/ubi-minimal:latest)
2019-03-02 10:36:29.978806894 -0600 CST container create d81e30f1310f
(image=registry.access.redhat.com/ubi8/ubi-init:latest)
```

関連情報

- **podman-events** の man ページ

第15章 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。保存したデータを使用して、再起動後に、チェックポイントの時点にコンテナーを復元できます。

15.1. ローカルでのコンテナーチェックポイントの作成および復元

以下の例は、要求ごとにインクリメントする整数を 1 つ返す Python ベースの Web サーバーをベースとしています。

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/ubi9/ubi

COPY counter.py /home/counter.py

RUN useradd -ms /bin/bash counter

RUN dnf -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**) の入力情報です。ビルドされたイメージはローカルに保存され、**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. コンテナーを復元します。

```
# 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** で開始あれるのではなく、以前の値で継続されます。

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。

関連情報

- Adding checkpoint/restore support to Podman

15.2. 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」のコンテナーを基にしています。

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を基に復元したコンテナーに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
```

同じチェックポイントから異なるコンテナーを使用して復元しているので、結果は、どの場合も **4** になる点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を作成し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。

関連情報

- Container migration with Podman on RHEL

15.3. 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別のシステムに移行する方法を示しています。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」のコンテナーを基にしています。

カウンター でタグ付けされたセクション。

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。理由は、Podman がローカルでコンテナーを利用できない場合に自動的にコンテナーをレジストリーから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ません。以前に構築およびタグ付けしたコンテナー (「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)をエクス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
```

このコンテナーの移行先のシステムには、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
```

これで、ステートフルコンテナーが、状態を失うことなく、別のシステムへ移行されました。

関連情報

- Container migration with Podman on RHEL

第16章 HPC 環境での 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。

16.1. PODMAN と MPI の使用

この例は、Open MPI から取得した ring.c プログラムを基にしています。この例では、全プロセスで値はリングのように渡されます。メッセージがランク 0 を渡すと必ず、値は 1 つ下がります。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 dnf 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** を統合するには、以下の定義でコンテナーを作成します。

```
$ cat Containerfile
FROM registry.access.redhat.com/ubi9/ubi

RUN dnf -y install openmpi-devel wget && \
    dnf 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。このコマンドは、CPU が 4 つあるシステムではコンテナーを 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、コンテナーごとに **ring** バイナリーのインスタンスを 1 台実行します。4 つプロセスはすべて、MPI 経由で相互に通信しています。

関連情報

- Podman in HPC environments

16.2. MPIRUN オプション

以下の **mpirun** オプションは、コンテナーの起動に使用します。

- **--mca orte_tmpdir_base /tmp/podman-mpirun** 行は、Open MPI に対し、**/tmp** ではなく、**/tmp/podman-mpirun** にその一時ファイルをすべて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には、別のノードではこのディレクトリーの名前は異なります。このような場合には、**/tmp** ディレクトリー全体をコンテナーにマウントする必要があり、操作がより複雑です。

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、コンテナーの起動に使用されます。

- **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、Open MPI が一時ディレクトリーとファイルをコンテナーで利用できるように、Podman にディレクトリーのマウントを指示します。
- **--userns=keep-id** 行を使用すると、コンテナー内外でのユーザー ID マッピングを保証します。
- **--net=host --pid=host --ipc=host** 行では、同じネットワーク、PID、および IPC 名前空間が設定されます。
- **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。

関連情報

- Podman in HPC environments

第17章 特殊なコンテナーイメージの実行

本章では、特殊なタイプのコンテナーイメージについて説明します。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、runlabelsと呼ばれるラベルが組み込まれています。**podman container runlabel <label>** コマンドでは、コンテナーイメージに対して **<label>** で定義されているコマンドを実行できます。サポートされるラベルは、**install**、**run**、および **uninstall** です。

17.1. ホストへの権限の付与

特権コンテナーと非特権コンテナーにはいくつかの違いがあります。たとえば、toolbox コンテナーは特権コンテナーです。以下は、コンテナーがホストに対して許可する、または許可しない権限の例です。

- **権限**:特権コンテナーは、コンテナーをホストから分離するセキュリティー機能を無効にします。特権コンテナーは、**podman run --privileged <image_name>** コマンドを使用して実行できます。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**:**podman run --privileged --pid=host <image_name>** コマンドを使用して、コンテナーでホストの PID 名前空間を使用できます。特権コンテナー内で **ps -e** コマンドを使用して、ホストで実行しているすべてのプロセスを一覧表示できます。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>**)。

- **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。**podman run --net=host <image_name>** コマンドを使用して、コンテナー内からホストの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。

- **プロセス間の通信**:ホストの IPC 機能は、特権コンテナーからアクセスできます。**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。

17.2. RUNLABEL が組み込まれたコンテナーイメージ

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、**podman**コマンドを使用してイメージの **<label>** で定義されたコマンドを実行できます。

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**:コンテナーの実行を終了した後にホストシステムをクリーンアップします。

17.3. ランレベルでの RSYSLOG の実行

rhel9/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** イメージには、**install**、**run** および **uninstall** の runlabelが含まれます。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/rhel{ProductNumberLink}/rsyslog
```

2. **rsyslog** の **install** runlabel を表示します。

```
# podman container runlabel install --display rhel{ProductNumberLink}/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel{ProductNumberLink}/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel{ProductNumberLink}/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** runlabel を実行します。

```
# podman container runlabel install rhel{ProductNumberLink}/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel{ProductNumberLink}/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel{ProductNumberLink}/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** runlabel を表示します。

```
# podman container runlabel run --display rhel{ProductNumberLink}/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/rhel{ProductNumberLink}/rsyslog:latest -e NAME=rsyslog --
restart=always registry.redhat.io/rhel{ProductNumberLink}/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** runlabel を実行します。

```
# podman container runlabel run rhel{ProductNumberLink}/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/rhel{ProductNumberLink}/rsyslog:latest -e NAME=rsyslog --
restart=always registry.redhat.io/rhel{ProductNumberLink}/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** runlabel を表示します。

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

7. **rsyslog** の **uninstall** runlabel を実行します。

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

注記

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ません。

第18章 CONTAINER-TOOLS API の使用

varlink ライブラリーを使用する Podman リモート API に代わって、新しい REST ベースの Podman 2.0 API が導入されました。新規 API は、ルートフル環境およびルートレス環境の両方で動作します。

Podman v2.0 RESTful API は、Podman および Docker 互換 API をサポートする Libpod API で構成されます。この新しい REST API を使用すると、cURL、Postman、Google の Advanced REST クライアントなどのプラットフォームから Podman を呼び出すことができます。

18.1. ROOT モードで SYSTEMD を使用した PODMAN API の有効化

この手順では、以下を行う方法を説明します。

1. systemd を使用して Podman API ソケットをアクティベートします。
2. Podman クライアントを使用して基本的なコマンドを実行します。

前提条件

- **podman-remote** パッケージがインストールされている。

```
# dnf install podman-remote
```

手順

1. サービスをすぐに起動します。

```
# systemctl enable --now podman.socket
```

2. **docker-podman** パッケージを使用して **var/lib/docker.sock** へのリンクを有効にするには以下を実行します。

```
# dnf install podman-docker
```

検証手順

1. Podman のシステム情報を表示します。

```
# podman-remote info
```

2. リンクを確認します。

```
# ls -al /var/run/docker.sock
lrwxrwxrwx. 1 root root 23 Nov  4 10:19 /var/run/docker.sock -> /run/podman/podman.sock
```

関連情報

- Podman v2.0 RESTful API
- A First Look At Podman 2.0 API
- スニークピーク: Podman の新しい REST API

18.2. ルートレスモードで SYSTEMD を使用した PODMAN API の有効化

この手順では、systemd を使用して Podman API ソケットおよび Podman API サービスをアクティベートする方法を説明します。

前提条件

- **podman-remote** パッケージがインストールされている。

```
# dnf install podman-remote
```

手順

1. サービスをすぐに有効にして起動します。

```
$ systemctl --user enable --now podman.socket
```

2. オプション。Docker を使用してプログラムがルートレス Podman ソケットを操作できるようにするには、以下を実行します。

```
$ export DOCKER_HOST=unix:///run/user/<uid>/podman//podman.sock
```

検証手順

1. ソケットのステータスを確認します。

```
$ systemctl --user status podman.socket
● podman.socket - Podman API Socket
 Loaded: loaded (/usr/lib/systemd/user/podman.socket; enabled; vendor preset: enabled)
Active: active (listening) since Mon 2021-08-23 10:37:25 CEST; 9min ago
Docs: man:podman-system-service(1)
Listen: /run/user/1000/podman/podman.sock (Stream)
CGroup: /user.slice/user-1000.slice/user@1000.service/podman.socket
```

podman.socket はアクティブで、**/run/user/<uid>/podman.podman.sock** をリッスンしています。**<uid>** はユーザーの ID です。

2. Podman のシステム情報を表示します。

```
$ podman-remote info
```

関連情報

- Podman v2.0 RESTful API
- A First Look At Podman 2.0 API
- スニークピーク: Podman の新しい REST API
- Exploring Podman RESTful API using Python and Bash

18.3. PODMAN API の手動実行

この手順では、Podman API の実行方法について説明します。手動での実行は、特に Docker の互換性レイヤーを使用する場合など、API 呼び出しのデバッグに便利です。

前提条件

- **podman-remote** パッケージがインストールされている。

```
# dnf install podman-remote
```

手順

1. REST API のサービスを実行します。

```
# podman system service -t 0 --log-level=debug
```

   - 値が 0 の場合はタイムアウトなしを意味します。ルートフルサービスのデフォルトのエンドポイントは **unix:/run/podman/podman.sock** です。
   - **--log-level <level>** オプションは、ロギングレベルを設定します。標準のロギングレベルは **debug**、**info**、**warn**、**error**、**fatal**、および **panic** です。

2. 別のターミナルで、Podman のシステム情報を表示します。**podman-remote** コマンドは、通常の **podman** コマンドとは異なり、Podman ソケットを介して通信します。

```
# podman-remote info
```

3. Podman API をトラブルシューティングし、要求および応答を表示するには、**curl** コマンドを使用します。JSON 形式で Linux サーバーの Podman インストールの情報を取得するには、以下を実行します。

```
# curl -s --unix-socket /run/podman/podman.sock http://d/v1.0.0/libpod/info | jq
    {
  "host": {
    "arch": "amd64",
    "buildahVersion": "1.15.0",
    "cgroupVersion": "v1",
    "conmon": {
      "package": "conmon-2.0.18-1.module+el8.3.0+7084+c16098dd.x86_64",
      "path": "/usr/bin/conmon",
      "version": "conmon version 2.0.18, commit:
7fd3f71a218f8d3a7202e464252aeb1e942d17eb"
    },
    …
  "version": {
    "APIVersion": 1,
    "Version": "2.0.0",
    "GoVersion": "go1.14.2",
    "GitCommit": "",
    "BuiltTime": "Thu Jan  1 01:00:00 1970",
    "Built": 0,
    "OsArch": "linux/amd64"
  }
}
```

jq ユーティリティーは、コマンドライン JSON プロセッサーです。

4. **registry.access.redhat.com/ubi8/ubi** コンテナーイメージをプルします。

```
# curl -XPOST --unix-socket /run/podman/podman.sock -v 'http://d/v1.0.0/images/create?
fromImage=registry.access.redhat.com%2Fubi8%2Fubi'
*   Trying /run/podman/podman.sock...
* Connected to d (/run/podman/podman.sock) port 80 (#0)
> POST /v1.0.0/images/create?fromImage=registry.access.redhat.com%2Fubi8%2Fubi
HTTP/1.1
> Host: d
> User-Agent: curl/7.61.1
> Accept: /
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Date: Tue, 20 Oct 2020 13:58:37 GMT
< Content-Length: 231
<
{"status":"pulling image () from registry.access.redhat.com/ubi8/ubi:latest,
registry.redhat.io/ubi8/ubi:latest","error":"","progress":"","progressDetail":
{},"id":"ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e"}
* Connection #0 to host d left intact
```

5. プルしたイメージを表示します。

```
# curl --unix-socket /run/podman/podman.sock -v 'http://d/v1.0.0/libpod/images/json' | jq
*   Trying /run/podman/podman.sock...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to d
(/run/podman/podman.sock) port 80 (0) > GET /v1.0.0/libpod/images/json HTTP/1.1 > Host: d
> User-Agent: curl/7.61.1 > Accept: / > < HTTP/1.1 200 OK < Content-Type: application/json
< Date: Tue, 20 Oct 2020 13:59:55 GMT < Transfer-Encoding: chunked < { [12498 bytes
data] 100 12485 0 12485 0 0 2032k 0 --:--:-- --:--:-- --:--:-- 2438k * Connection #0 to host d
left intact [ { "Id":
"ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e",
"RepoTags": [ "registry.access.redhat.com/ubi8/ubi:latest", "registry.redhat.io/ubi8/ubi:latest"
], "Created": "2020-09-01T19:44:12.470032Z", "Size": 210838671, "Labels": { "architecture":
"x86_64", "build-date": "2020-09-01T19:43:46.041620", "com.redhat.build-host": "cpt-
1008.osbs.prod.upshift.rdu2.redhat.com", ... "maintainer": "Red Hat, Inc.", "name": "ubi8", ...
"summary": "Provides the latest release of Red Hat Universal Base Image 8.", "url":
"https://access.redhat.com/containers//registry.access.redhat.com/ubi8/images/8.2-347",
      ...
    },
    "Names": [
      "registry.access.redhat.com/ubi8/ubi:latest",
      "registry.redhat.io/ubi8/ubi:latest"
    ],
    ...
    ]
  }
]
```

関連情報

- Podman v2.0 RESTful API
- スニークピーク: Podman の新しい REST API
- Exploring Podman RESTful API using Python and Bash
- **podman-system-service** の man ページ