Red Hat Enterprise Linux 9

Image Mode for RHEL を使用したオペレーティングシステムの構築、デプロイ、管理

Red Hat Enterprise Linux 9 で RHEL のブート可能なコンテナーイメージを使用する

Last Updated: 2024-06-27

Red Hat Enterprise Linux 9 Image Mode for RHEL を使用したオペレーティングシステムの構築、デプロイ、管理

Red Hat Enterprise Linux 9 で RHEL のブート可能なコンテナーイメージを使用する

法律上の通知

概要

RHEL のブート可能なコンテナーイメージを使用すると、他のコンテナーと同じようにオペレーティングシステムを構築、デプロイ、管理できます。単一のコンテナーネイティブワークフローに集約して、アプリケーションから基盤となる OS まですべてを管理できます。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

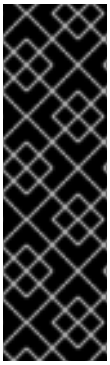
Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **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追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 IMAGE MODE FOR RHEL の概要

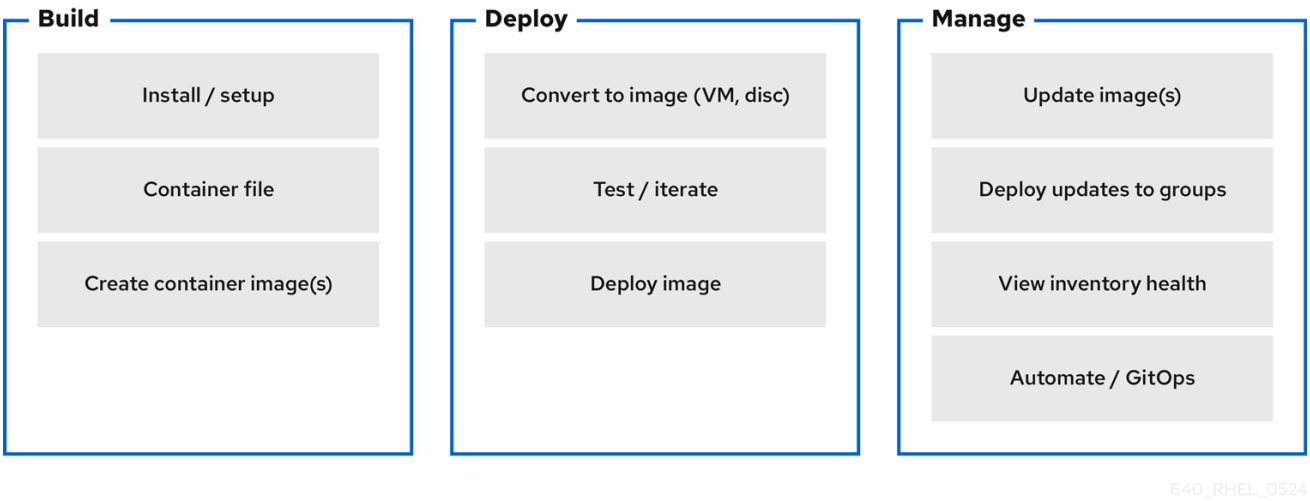
Image Mode for RHEL を使用すると、アプリケーションコンテナーと同じツールと手法を使用して、オペレーティングシステムを構築、テスト、およびデプロイできます。Image Mode for RHEL は、ブート可能なコンテナーイメージ **registry.redhat.io/rhel9/rhel-bootc** を使用することで利用できます。RHEL のブート可能なコンテナーイメージは、従来は除外されていた起動に必要な追加コンポーネント (カーネル、initrd、ブートローダー、ファームウェアなど) が含まれている点で、既存のアプリケーションの Universal Base Images (UBI) とは異なります。



重要

Red Hat は、**rhel9/rhel-bootc** コンテナーイメージをテクノロジープレビューとして提供します。テクノロジープレビュー機能は、近々発表予定の製品イノベーションをリリースに先駆けて提供します。これにより、お客様は機能をテストし、開発プロセス中にフィードバックを提供することができます。ただし、これらの機能は完全にはサポートされません。テクノロジープレビュー機能のドキュメントは不完全であったり、基本的なインストールや設定に関する情報しか含まれていなかったりする場合があります。テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

図1.1 Image Mode for RHEL を使用したオペレーティングシステムの構築、デプロイ、管理



Red Hat は、次のコンピューターアーキテクチャー用のブート可能なコンテナーイメージを提供します。

- AMD および Intel 64 ビットアーキテクチャー (x86-64-v2)
- 64 ビット ARM アーキテクチャー (ARMv8.0-A)

Image Mode for RHEL の利点は、システムのライフサイクル全体にわたって得られます。次のリストには、いくつかの最も重要な利点が含まれます。

コンテナーイメージは他のイメージ形式よりも簡単に理解して使用でき、ビルドが高速

Containerfile (Dockerfile とも呼ばれます) は、イメージのコンテンツとビルド手順を定義する単純なアプローチを提供します。コンテナーイメージは、しばしば他のイメージ作成ツールよりもビルドとイテレートが非常に高速です。

プロセス、インフラストラクチャー、リリースアーティファクトを統合

アプリケーションをコンテナーとして配布すると、同じインフラストラクチャーとプロセスを使用して、基盤となるオペレーティングシステムを管理できます。

イミュータブルな更新

Image Mode for RHEL では、コンテナー化されたアプリケーションがイミュータブルな方法で更新されるのと同様に、オペレーティングシステムもイミュータブルな方法で更新されます。**rpm-ostree** システムを使用する場合と同じように、更新を起動し、必要に応じてロールバックできます。

警告

rpm-ostree を使用して変更を加えたり、コンテンツをインストールしたりすることはサポートされていません。

ハイブリッドクラウド環境間での移植性

ブート可能なコンテナーイメージは、物理環境、仮想化環境、クラウド環境、エッジ環境全体で使用できます。

コンテナーはイメージをビルド、トランスポート、実行するための基盤を提供します。ただし、これらのブート可能なコンテナーイメージをインストールメカニズムでデプロイするか、またはディスクイメージに変換した後は、システムはコンテナーとして実行されないことを理解することが重要です。

サポートされるイメージタイプ は次のとおりです。

- コンテナーイメージ形式: OCI
- ディスクイメージ形式:
  - ISO
  - QEMU copy-on-write (QCOW2)、Raw
  - Amazon Machine Image (AMI)
  - 仮想マシンイメージ (VMI)
  - 仮想マシンディスク (VMDK)

コンテナーは、以下の手段を提供することで、RHEL システムのライフサイクルを合理化します。

コンテナーイメージのビルド

Containerfile を変更することで、ビルド時にオペレーティングシステムを設定できます。Image Mode for RHEL は、コンテナーイメージ **registry.redhat.io/rhel9/rhel-bootc** を使用することで利用できます。Podman、OpenShift Container Platform、またはその他の標準のコンテナービルドツールを使用して、コンテナーとコンテナーイメージを管理できます。CI/CD パイプラインを使用してビルドプロセスを自動化できます。

コンテナーイメージのバージョン管理、ミラーリング、テスト

Podman や OpenShift Container Platform などのコンテナーツールを使用して、派生したブート可能なコンテナーイメージのバージョン管理、ミラーリング、イントロスペクション、署名を行うことができます。

コンテナーイメージのターゲット環境へのデプロイ

イメージをデプロイする方法はいくつか存在します。

- **Anaconda**: RHEL で使用されるインストールプログラムです。Anaconda とキックスタートを使用してインストールプロセスを自動化することで、すべてのイメージタイプをターゲット環境にデプロイできます。
- **bootc-image-builder**: コンテナーイメージをさまざまな種類のディスクイメージに変換し、必要に応じてイメージレジストリーまたはオブジェクトストレージにアップロードする、コンテナー化されたツールです。
- **bootc**: コンテナーレジストリーからコンテナーイメージを取得してシステムにインストールしたり、オペレーティングシステムを更新したり、既存の ostree ベースのシステムから切り替えたりするツールです。RHEL のブート可能なコンテナーイメージには、デフォルトで **bootc** ユーティリティーが含まれており、すべてのイメージタイプで動作します。ただし、**rpm-ostree** はサポートされていないため、変更を加えるために使用してはならないことに注意してください。

オペレーティングシステムの更新

システムは、デプロイメント後にロールバックが可能なインプレーストランザクション更新をサポートします。自動更新はデフォルトでオンになっています。systemd サービスユニットと systemd タイマーユニットファイルがコンテナーレジストリーの更新をチェックし、システムに適用します。更新はトランザクショナルであるため、再起動が必要です。より高度なロールアウトやスケジュールされたロールアウトが必要な環境では、自動更新を無効にし、**bootc** ユーティリティーを使用してオペレーティングシステムを更新してください。

RHEL には 2 つのデプロイメントモードがあります。どちらも、デプロイメント時の安定性、信頼性、パフォーマンスは同様です。

1. **パッケージモード**: オペレーティングシステムは RPM パッケージを使用し、**dnf** パッケージマネージャーを使用して更新されます。ルートファイルシステムはミュータブルです。
2. **イメージモード**: RHEL をビルド、デプロイ、管理するためのコンテナーネイティブなアプローチです。同じ RPM パッケージがベースイメージとして配信され、更新はコンテナーイメージとしてデプロイされます。ルートファイルシステムは、**/etc** と **/var** を除きデフォルトではイミュータブルであり、ほとんどのコンテンツはコンテナーイメージから取得されます。

どちらのデプロイメントモードを使用しても、他のコンテナーアプリケーションと同じ方法でオペレーティングシステムをビルド、テスト、共有、デプロイ、管理できます。

1.1. 前提条件

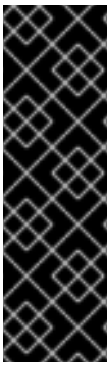
- RHEL 9 システムをサブスクライブしている。詳細は、RHEL システム登録のスタートガイドドキュメントを参照してください。
- コンテナーレジストリーがある。レジストリーをローカルに作成することも、Quay.io サービスで無料アカウントを作成することもできます。Quay.io アカウントを作成するには、Red Hat Quay.io ページを参照してください。
- 実稼働用または開発者サブスクリプションを持つ Red Hat アカウントがある。無料の開発者サブスクリプションは、Red Hat Enterprise Linux Overview ページで入手できます。
- registry.redhat.io に対して認証済みである。詳細は、Red Hat コンテナーレジストリーの認証の記事を参照してください。

1.2. 関連情報

- Introducing image mode for RHEL and bootable containers in Podman Desktop クイックスタートガイド
- Image mode for Red Hat Enterprise Linux quick start: AI inference クイックスタートガイド
- ブログ記事 Getting Started with Podman AI Lab
- 製品ドキュメント Anaconda のカスタマイズ
- 製品ドキュメント 高度な RHEL 9 インストールの実行 (キックスタート)
- 製品ドキュメント RHEL システムイメージのカスタマイズ
- 製品ドキュメント RHEL for Edge イメージの作成、インストール、および管理

第2章 RHEL のブート可能なコンテナーイメージのビルドおよびテスト

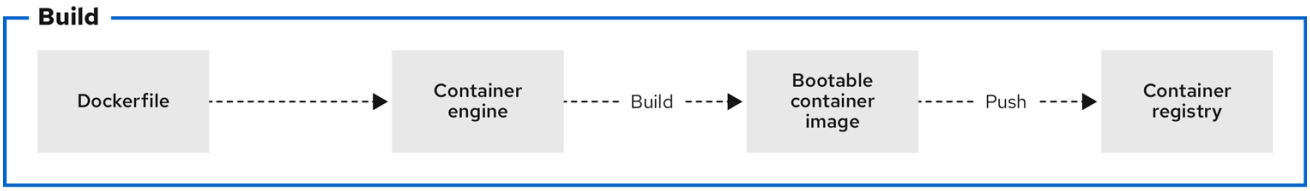
次の手順では、Podman を使用してコンテナーイメージをビルドおよびテストします。OpenShift Container Platform などの他のツールを使用することもできます。コンテナーを使用して RHEL システムを設定する例については、rhel-bootc-examples リポジトリーを参照してください。



重要

Red Hat は、**rhel9/rhel-bootc** コンテナーイメージをテクノロジープレビューとして提供します。テクノロジープレビュー機能は、近々発表予定の製品イノベーションをリリースに先駆けて提供します。これにより、お客様は機能をテストし、開発プロセス中にフィードバックを提供することができます。ただし、これらの機能は完全にはサポートされません。テクノロジープレビュー機能のドキュメントは不完全であったり、基本的なインストールや設定に関する情報しか含まれていなかったりする場合があります。テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

図2.1 Containerfile からの指示を使用してイメージを構築し、コンテナーをテストし、イメージをレジストリーにプッシュして他のユーザーと共有する



639_RHEL_0524

一般的な **Containerfile** の構造は次のとおりです。

```
FROM registry.redhat.io/rhel9/rhel-bootc:latest

RUN dnf -y install [software] [dependencies] && dnf clean all

ADD [application]
ADD [configuration files]

RUN [config scripts]
```

rhel-9-bootc コンテナーイメージは OCI イメージ形式を再利用します。

- **rhel-9-bootc** コンテナーイメージは、システムへのインストール時にはコンテナー設定セクション (**Config**) を無視します。
- **rhel-9-bootc** コンテナーイメージは、**podman** や **docker** などのコンテナーランタイムを使用してこのイメージを実行するときには、コンテナー設定セクション (**Config**) を無視しません。

たとえば、**rhel-9-bootc** イメージのシステムへのインストール時には、**Containerfile** 内の以下のコマンドは無視されます。

- **ENTRYPOINT** および **CMD** (OCI: **Entrypoint**/**Cmd**): 代わりに **CMD**/**sbin**/**init** を設定できます。
- **ENV** (OCI: **Env**): **systemd** 設定を変更して、グローバルシステム環境を設定します。

- **EXPOSE** (OCI: **exposedPorts**): ランタイム時にシステムのファイアウォールやネットワークがどのように機能するかとは無関係です。
- **USER** (OCI: **User**): 代わりに、RHEL ブート可能コンテナー内の個々のサービスを、権限のないユーザーとして実行するように設定します。

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。

注記

このリリースでは、カスタム **rhel-bootc** ベースイメージのビルドはサポートされていません。

2.1. コンテナーイメージのビルド

Containerfile からの指示を使用してイメージをビルドするには、**podman build** コマンドを使用します。

前提条件

- **container-tools** メタパッケージがインストールされている。

手順

1. **Containerfile** を作成します。

```
$ cat Containerfile
FROM registry.redhat.io/rhel9/rhel-bootc:latest
RUN dnf -y install cloud-init && \
    ln -s ../cloud-init.target /usr/lib/systemd/system/default.target.wants && \
    dnf clean all
```

この **Containerfile** の例では、**cloud-init** ツールが追加されています。そのため、SSH キーを自動的に取得し、インフラストラクチャーからスクリプトを実行できるほか、インスタンスメタデータから設定とシークレットを収集することもできます。たとえば、このコンテナーイメージを、事前に生成した AWS または KVM ゲストシステムに使用できます。

2. 現在のディレクトリーの **Containerfile** を使用して、**<image>** イメージをビルドします。

```
$ podman build -t quay.io/<namespace>/<image>:<tag>
```

検証

- すべてのイメージをリスト表示します。

```
$ podman images
REPOSITORY                  TAG     IMAGE ID      CREATED             SIZE
localhost/<image>           latest  b28cd00741b3  About a minute ago  2.1 GB
```

関連情報

- コンテナーレジストリーの使用

- Buildah を使用した Containerfile からのイメージのビルド

2.2. コンテナーイメージの実行

podman run コマンドを使用して、コンテナーを実行およびテストします。

前提条件

- **container-tools** メタパッケージがインストールされている。

手順

- **quay.io/<namespace>/<image>:<tag>** コンテナーイメージに基づいて、**mybootc** という名前のコンテナーを実行します。

```
$ podman run -it --rm --name mybootc quay.io/<namespace>/<image>:<tag> /bin/bash
```

  - **-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。
  - **--rm** オプションは、コンテナーの終了後に **quay.io/<namespace>/<image>:<tag>** コンテナーイメージを削除します。

検証

- 実行中のすべてのコンテナーをリスト表示します。

```
$ podman ps
CONTAINER ID  IMAGE                                COMMAND         CREATED        STATUS
PORTS   NAMES
7ccd6001166e  quay.io/<namespace>/<image>:<tag>  /sbin/init  6 seconds ago  Up 5
seconds ago          mybootc
```

関連情報

- podman run コマンド

2.3. コンテナーイメージのレジストリーへのプッシュ

podman push コマンドを使用して、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し、他のユーザーと共有します。次の手順では、Red Hat Quay レジストリーを使用します。

前提条件

- **container-tools** メタパッケージがインストールされている。
- イメージがビルドされ、ローカルシステムで使用できる。

- Red Hat Quay レジストリーを作成した。詳細は、概念実証 - Red Hat Quay のデプロイ を参照してください。

手順

- ローカルストレージからレジストリーに **quay.io/<namespace>/<image>:<tag>** コンテナーイメージをプッシュします。

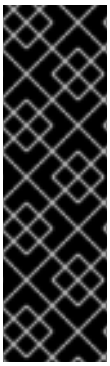
```
$ podman push quay.io/<namespace>/<image>:<tag>
```

関連情報

- UBI イメージの再配布

第3章 BOOTC-IMAGE-BUILDER を使用した BOOTC 互換ベースディスクイメージの作成

テクノロジープレビューとして利用可能な **bootc-image-builder** は、ブート可能なコンテナーイメージからディスクイメージを作成するためのコンテナー化されたツールです。ビルドしたイメージを使用して、エッジ、サーバー、クラウドなどのさまざまな環境にディスクイメージをデプロイできます。



重要

Red Hat は、**bootc-image-builder** ツールをテクノロジープレビューとして提供します。テクノロジープレビュー機能は、近々発表予定の製品イノベーションをリリースに先駆けて提供します。これにより、お客様は機能をテストし、開発プロセス中にフィードバックを提供することができます。ただし、これらの機能は完全にはサポートされません。テクノロジープレビュー機能のドキュメントは不完全であったり、基本的なインストールや設定に関する情報しか含まれていなかったりする場合があります。テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

3.1. BOOTC-IMAGE-BUILDER の RHEL 用イメージモードの導入

bootc-image-builder ツールを使用すると、ブート可能なコンテナーイメージをさまざまなプラットフォームや形式のディスクイメージに変換できます。ブート可能なコンテナーイメージをディスクイメージに変換することは、ブート可能なコンテナーをインストールすることと同じです。これらのディスクイメージは、ターゲット環境にデプロイした後、コンテナーレジストリーから直接更新できます。

注記

このリリースでは、プライベートレジストリーから取得されるベースディスクイメージを **bootc-image-builder** を使用してビルドすることはサポートされていません。

bootc-image-builder tool は、次のイメージタイプの生成をサポートします。

- 非接続インストールに適したディスクイメージ形式 (ISO など)
- 以下のような仮想ディスクイメージ形式:
  - QEMU copy-on-write (QCOW2)
  - Amazon Machine Image (AMI) - Raw
  - 仮想マシンイメージ (VMI)

コンテナーイメージからデプロイすると同じインストール結果を得られるため、仮想マシンまたはサーバーを実行するときに便利です。同じコンテナーイメージからビルドする場合、当該一貫性は複数の異なるイメージタイプとプラットフォームにわたって保持されます。その結果、プラットフォーム間でオペレーティングシステムイメージを維持するための労力を最小限に抑えることができます。また、**bootc-image-builder** を使用して新しいディスクイメージを再作成およびアップロードする代わりに、**bootc** ツールを使用して、これらのディスクイメージからデプロイしたシステムを更新することもできます。

注記

汎用ベースコンテナーイメージには、デフォルトのパスワードや SSH キーは含まれません。また、**bootc-image-builder** ツールを使用して作成したディスクイメージには、**cloud-init** などの一般的なディスクイメージで使用できるツールは含まれません。これらのディスクイメージは、変換されたコンテナーイメージのみで構成されます。

rhel-9-bootc イメージを直接デプロイすることもできますが、このブート可能なベースイメージから派生した独自のカスタマイズイメージを作成することもできます。**bootc-image-builder** ツールは、**rhel-9-bootc** OCI コンテナーイメージを入力として受け取ります。

関連情報

- cloud-init を使用する Red Hat 製品

3.2. BOOTC-IMAGE-BUILDER のインストール

bootc-image-builder はコンテナーとしての使用が意図されており、RHEL で RPM パッケージとして利用することはできません。アクセスするには、以下の手順に従います。

前提条件

- **container-tools** メタパッケージがインストールされている。メタパッケージには、Podman、Buildah、Skopeo などのすべてのコンテナーツールが含まれます。
- **registry.redhat.io** に対して認証されている。詳細は、Red Hat コンテナーレジストリーの認証 を参照してください。

手順

1. ログインして、**registry.redhat.io** に対する認証を行います。

```
$ sudo podman login registry.redhat.io
```

2. **bootc-image-builder** ツールをインストールします。

```
$ sudo podman pull registry.redhat.io/rhel9/bootc-image-builder
```

検証

- ローカルシステムにプルしたすべてのイメージをリスト表示します。

```
$ sudo podman images
REPOSITORY                                  TAG         IMAGE ID      CREATED       SIZE
registry.redhat.io/rhel9/bootc-image-builder  latest      b361f3e845ea  24 hours ago  676 MB
```

関連情報

- Red Hat コンテナーレジストリーの認証
- レジストリーからのイメージの取得 (プル)

3.3. BOOTC-IMAGE-BUILDER を使用した QCOW2 イメージの作成

コマンドを実行しているアーキテクチャーの QEMU ディスクイメージ (QCOW2) のイメージに、RHEL ブート可能コンテナーイメージをビルドします。

RHEL ベースイメージにはデフォルトのユーザーは含まれません。必要に応じて、**--config** オプションを使用してユーザー設定を注入し、bootc-image-builder コンテナーを実行できます。あるいは、**cloud-init** を使用してベースイメージを設定し、初回起動時にユーザーと SSH キーを注入することもできます。cloud-init を使用したユーザーと SSH キーの注入 を参照してください。

前提条件

- ホストマシンに Podman がインストールされている。
- ホストマシンに **virt-install** がインストールされている。
- **bootc-image-builder** ツールを実行し、コンテナーを **--privileged** モードで実行して、イメージをビルドするための root アクセスがある。

手順

1. オプション: ユーザーアクセスを設定するための **config.toml** を作成します。次に例を示します。

```
[[blueprint.customizations.user]]
name = "user"
password = "pass"
key = "ssh-rsa AAA ... user@email.com"
groups = ["wheel"]
```

2. **bootc-image-builder** を実行します。必要に応じて、ユーザーアクセス設定を使用する場合は、**config.toml** を引数として渡します。

注記

コンテナーストレージマウントおよび **--local** イメージオプションがない場合は、イメージをパブリックにする必要があります。

a. 以下は、パブリック QCOW2 イメージを作成する例です。

```
$ sudo podman run \
    --rm \
    -it \
    --privileged \
    --pull=newer \
    --security-opt label=type:unconfined_t \
    -v ./config.toml:/config.toml \
    -v ./output:/output \
    -v /var/lib/containers/storage:/var/lib/containers/storage \
registry.redhat.io/rhel9/bootc-image-builder:latest \
    --type qcow2 \
    --config config.toml \
  quay.io/<namespace>/<image>:<tag>
```

b. 以下は、プライベート QCOW2 イメージを作成する例です。

```
$ sudo podman run \
    --rm \
    -it \
    --privileged \
    --pull=newer \
    --security-opt label=type:unconfined_t \
    -v ./config.toml:/config.toml \
    -v ./output:/output \
    registry.redhat.io/rhel9/bootc-image-builder:latest \
    --type qcow2 \
    --config config.toml \
   quay.io/<namespace>/<image>:<tag>
```

.qcow2 イメージは出力フォルダーにあります。

次のステップ

- イメージをデプロイできます。QCOW2 ディスクイメージを使用した KVM でのコンテナーイメージのデプロイ を参照してください。
- イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

3.4. BOOTC-IMAGE-BUILDER を使用した AMI イメージの作成および AWS へのアップロード

ブート可能なコンテナーイメージから Amazon Machine Image (AMI) を作成し、それを使用して Amazon Web Services EC2 (Amazon Elastic Compute Cloud) インスタンスを起動します。

前提条件

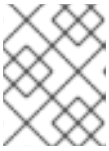
- ホストマシンに Podman がインストールされている。
- AWS アカウント内に既存の **AWS S3** バケットがある。
- **bootc-image-builder** ツールを実行し、コンテナーを **--privileged** モードで実行して、イメージをビルドするための root アクセスがある。
- AMI を AWS アカウントにインポートするために、アカウントに **vmimport** サービスロールが設定されている。

手順

1. ブート可能なコンテナーイメージからディスクイメージを作成します。
   - Containerfile でユーザーの詳細を設定します。必ず sudo アクセスを割り当ててください。
   - Containerfile で設定したユーザーを使用して、カスタマイズされたオペレーティングシステムイメージをビルドします。これにより、パスワードなしの sudo アクセスを持つデフォルトのユーザーが作成されます。
2. オプション: **cloud-init** を使用してマシンイメージを設定します。cloud-init を使用したユーザーと SSH キーの注入 を参照してください。以下に例を示します。

```
FROM registry.redhat.io/rhel9/rhel-bootc:9.4

RUN dnf -y install cloud-init && \
    ln -s ../cloud-init.target /usr/lib/systemd/system/default.target.wants && \
    rm -rf /var/{cache,log} /var/lib/{dnf,rhsm}
```



注記

cloud-init により、インスタンスメタデータを使用してユーザーや設定を追加することもできます。

3. ブート可能なコンテナーイメージをビルドします。たとえば、イメージを **x86_64** AWS マシンにデプロイするには、次のコマンドを使用します。

```
$ podman build -t quay.io/<namespace>/<image>:<tag> .
$ podman push quay.io/<namespace>/<image>:<tag> .
```

4. **bootc-image-builder** ツールを使用して、bootc コンテナーイメージから AMI を作成します。

```
$ sudo podman run \
  --rm \
  -it \
  --privileged \
  --pull=newer \
  -v $HOME/.aws:/root/.aws:ro \
  --env AWS_PROFILE=default \
  registry.redhat.io/rhel9/bootc-image-builder:latest \
  --type ami \
  --aws-ami-name rhel-bootc-x86 \
  --aws-bucket rhel-bootc-bucket \
  --aws-region us-east-1 \
quay.io/<namespace>/<image>:<tag>
```

注記

以下のフラグはすべてまとめて指定する必要があります。フラグを指定しない場合、AMI は出力ディレクトリーにエクスポートされます。

- **--aws-ami-name** - AWS の AMI イメージの名前
- **--aws-bucket** - AMI を作成する際の中間ストレージのターゲット S3 バケット名
- **--aws-region** - AWS アップロードのターゲットリージョン
  bootc-image-builder ツールは AMI イメージをビルドし、ビルド後に AWS 認証情報を使用して AMI イメージをプッシュおよび登録することで、AWS s3 バケットにアップロードします。

次のステップ

- イメージをデプロイできます。AMI ディスクイメージを使用したコンテナーイメージの AWS へのデプロイ を参照してください。

- イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

関連情報

- AWS CLI のドキュメント

3.5. BOOTC-IMAGE-BUILDER を使用した RAW ディスクイメージの作成

bootc-image-builder を使用すると、ブート可能なコンテナーイメージを、MBR または GPT パーティションテーブルを持つ Raw イメージに変換できます。RHEL ベースイメージにはデフォルトのユーザーを含みません。そのため、必要に応じて **--config** オプションを使用してユーザー設定を注入し、**bootc-image-builder** コンテナーを実行することもできます。あるいは、**cloud-init** を使用してベースイメージを設定し、初回起動時にユーザーと SSH キーを注入することもできます。cloud-init を使用したユーザーと SSH キーの注入 を参照してください。

前提条件

- ホストマシンに Podman がインストールされている。
- **bootc-image-builder** ツールを実行し、コンテナーを **--privileged** モードで実行して、イメージをビルドするための root アクセスがある。
- コンテナーストレージにターゲットコンテナーイメージをプルした。

手順

1. オプション: ユーザーアクセスを設定するための **config.toml** を作成します。次に例を示します。

```
[[blueprint.customizations.user]]
name = "user"
password = "pass"
key = "ssh-rsa AAA ... user@email.com"
groups = ["wheel"]
```

2. **bootc-image-builder** を実行します。ユーザーアクセス設定を使用する場合は、**config.toml** を引数として渡します。

```
$ sudo podman run \
    --rm \
    -it \
    --privileged \
    --pull=newer \
    --security-opt label=type:unconfined_t \
    -v /var/lib/containers/storage:/var/lib/containers/storage \
    -v ./config.toml:/config.toml \
    -v ./output:/output \
    registry.redhat.io/rhel9/bootc-image-builder:latest \
    --local \
    --type raw \
    --config config.toml \
   quay.io/<namespace>/<image>:<tag>
```

.raw イメージは出力フォルダーにあります。

次のステップ

- イメージをデプロイできます。QCOW2 ディスクイメージを使用した KVM でのコンテナーイメージのデプロイ を参照してください。
- イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

3.6. BOOTC-IMAGE-BUILDER を使用した ISO イメージの作成

bootc-image-builder を使用すると、ブート可能なコンテナーのオフラインデプロイメントを実行できる ISO を作成できます。

前提条件

- ホストマシンに Podman がインストールされている。
- **bootc-image-builder** ツールを実行し、コンテナーを **--privileged** モードで実行して、イメージをビルドするための root アクセスがある。

手順

- **bootc-image-builder** を実行します。

```
$ sudo podman run \
    --rm \
    -it \
    --privileged \
    --pull=newer \
    --security-opt label=type:unconfined_t \
    -v $(pwd)/config.toml:/config.toml \
    -v $(pwd)/output:/output \
    registry.redhat.io/rhel9/bootc-image-builder:latest \
    --type iso \
    --config config.toml \
  quay.io/<namespace>/<image>:<tag>
```

.iso イメージは出力フォルダーにあります。

次のステップ

- ISO イメージは、USB スティックや Install-on-boot などの無人インストール方法で使用できます。インストール可能なブート ISO には、設定済みのキックスタートファイルが含まれます。Anaconda とキックスタートを使用したコンテナーイメージのデプロイ を参照してください。

警告

キックスタートはシステム上の最初のディスクを自動的に再フォーマットするように設定されています。そのため、既存のオペレーティングシステムまたはデータを有するマシンで ISO を起動すると、破壊的な結果を招く可能性があります。

- イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

3.7. 検証とトラブルシューティング

AWS イメージの要件の設定に問題がある場合は、次のドキュメントを参照してください。

- AWS IAM account manager
- Using high-level (s3) commands with the AWS CLI
- S3 buckets
- Regions and Zones
- カスタマイズした RHEL イメージの AWS での起動

ユーザー、グループ、SSH キー、シークレットの詳細は、以下を参照してください。

- Image Mode for RHEL でのユーザー、グループ、SSH キー、シークレットの管理

第4章 RHEL のブート可能なイメージのデプロイ

次のさまざまなメカニズムを使用して、**rhel-bootc** コンテナーイメージをデプロイできます。

- Anaconda
- **bootc-image-builder**
- **bootc install**

使用できるブート可能なイメージのタイプは次のとおりです。

- 次のような **bootc image-builder** を使用して生成したディスクイメージ:
  - QCOW2 (QEMU copy-on-write、仮想ディスク)
  - Raw (Mac 形式)
  - AMI (Amazon Cloud)
  - ISO: USB スティックまたは Install-on-boot を使用した無人インストール方法

デプロイ可能なレイヤーイメージを作成した後に、そのイメージをホストにインストールする方法はいくつかあります。

- 次のメカニズムを使用することで、RHEL インストーラーとキックスタートを使用してレイヤーイメージをベアメタルシステムにインストールできます。
  - USB を使用したデプロイ
  - PXE
- **bootc-image-builder** を使用してコンテナーイメージをブート可能なイメージに変換し、ベアメタルまたはクラウド環境にデプロイすることもできます。

インストール方法は 1 回だけ実行されます。イメージをデプロイすると、その後の更新は、更新の公開時にコンテナーレジストリーから直接適用されます。

図4.1 基本ビルドインストーラー bootc install を使用して起動可能なコンテナーイメージをデプロイするか、Anaconda と Kickstart を使用してコンテナーイメージをデプロイする

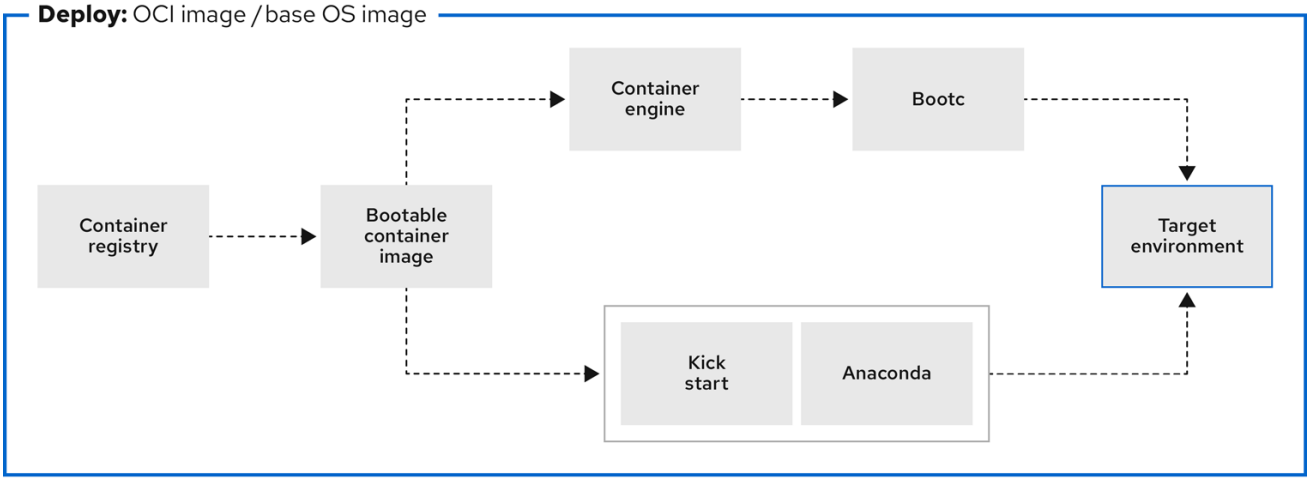
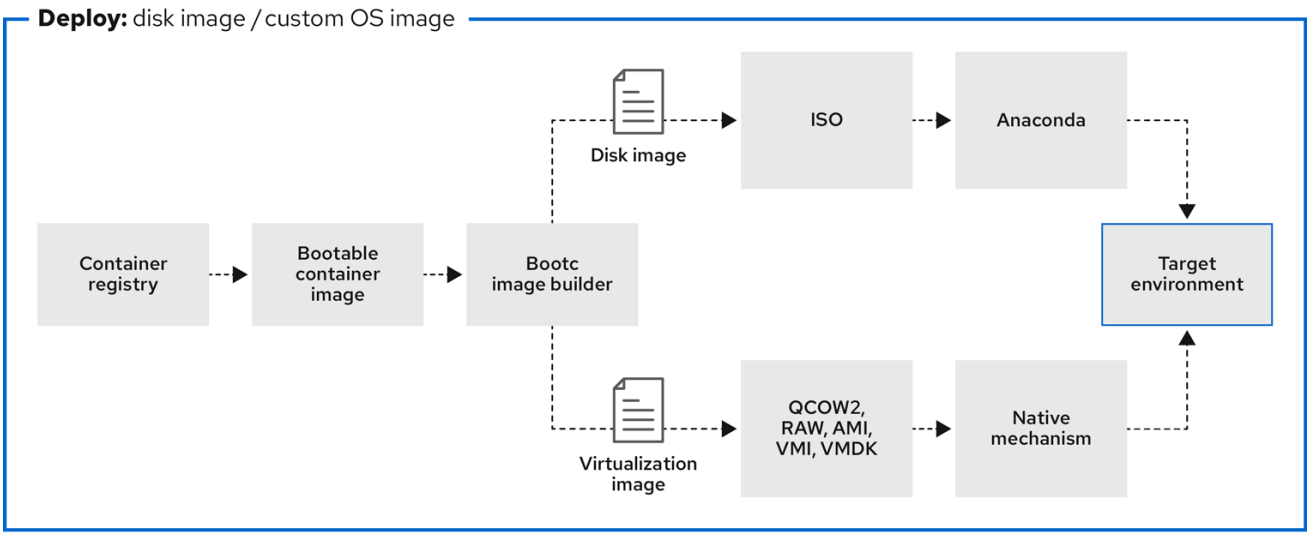


図4.2 bootc-image-builder を使用して起動可能なコンテナーイメージからディスクイメージを作成し、Anaconda、bootc-image-builder、または bootc install を使用してエッジ、サーバー、クラウドなどのさまざまな環境にディスクイメージをデプロイする



4.1. QCOW2 ディスクイメージを使用した KVM でのコンテナーイメージのデプロイ

bootc-image-builder ツールを使用して RHEL のブート可能なコンテナーイメージから QEMU ディスクイメージを作成したら、仮想化ソフトウェアを使用して起動できます。

前提条件

- コンテナーイメージを作成した。bootc-image-builder を使用した QCOW2 イメージの作成 を参照してください。
- コンテナーイメージをアクセス可能なリポジトリーにプッシュした。

手順

- **libvirt** を使用して作成したコンテナーイメージを実行します。詳細は、コマンドラインインターフェイスを使用した仮想マシンの作成 を参照してください。
  - 次の例では **libvirt** を使用しています。

```
$ sudo virt-install \
  --name bootc \
  --memory 4096 \
  --vcpus 2 \
  --disk qcow2/disk.qcow2 \
  --import \
  --os-variant rhel9-unknown
```

検証

- コンテナーイメージを実行している仮想マシンに接続します。詳細は、仮想マシンへの接続 を参照してください。

次のステップ

- イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

関連情報

- 仮想化の設定および管理

4.2. AMI ディスクイメージを使用したコンテナーイメージの AWS へのデプロイ

bootc-image-builder ツールを使用してブート可能なコンテナーイメージから AMI を作成し、それを AWS s3 バケットにアップロードしたら、AMI ディスクイメージを使用してコンテナーイメージを AWS にデプロイできます。

前提条件

- ブート可能なコンテナーイメージから Amazon Machine Image (AMI) を作成した。bootc-image-builder を使用した AMI イメージの作成および AWS へのアップロード を参照してください。
- 以前に作成した Containerfile で **cloud-init** が使用でき、ユースケースに合わせたレイヤーイメージを作成できる。

手順

1. ブラウザーで Service→EC2 にアクセスし、ログインします。
2. AWS コンソールのダッシュボードメニューで、正しいリージョンを選択します。イメージが正常にアップロードされたことを示すには、イメージが **Available** ステータスになっている必要があります。
3. AWS ダッシュボードでイメージを選択し、**Launch** をクリックします。
4. 新しく開いたウィンドウで、イメージを起動するために必要なリソースに応じて、インスタンスタイプを選択します。**Review and Launch** をクリックします。
5. インスタンスの詳細を確認します。変更が必要な場合は、各セクションを編集できます。**Launch** をクリックします。
6. インスタンスを起動する前に、インスタンスにアクセスするための公開鍵を選択します。既存のキーペアを使用するか、キーペアーを新規作成します。
7. インスタンスを起動するには、**Launch Instance** をクリックします。**Initializing** と表示されるインスタンスのステータスを確認できます。
   インスタンスのステータスが **Running** になると、**Connect** ボタンが有効になります。
8. **Connect** をクリックします。ウィンドウが表示され、SSH を使用して接続する方法の説明が表示されます。
9. 次のコマンドを実行して、自分だけが読み取れるように秘密鍵ファイルのパーミッションを設定します。Connect to your Linux instance を参照してください。

```
$ chmod 400 <your-instance-name.pem>
```

10. パブリック DNS を使用してインスタンスに接続します。

```
$ ssh -i <your-instance-name.pem>ec2-user@<your-instance-IP-address>
```

注記

インスタンスは停止しない限り実行され続けます。

検証

イメージを起動した後、次の操作を実行できます。

- ブラウザーで http://<your_instance_ip_address> への接続を試行します。
- SSH でインスタンスに接続している間にアクションが実行できるかどうかを確認します。

次のステップ

- イメージをデプロイした後、イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

関連情報

- イメージの AWS Cloud AMI へのプッシュ
- Amazon Machine Images (AMI)

4.3. ANACONDA とキックスタートを使用したコンテナーイメージのデプロイ

bootc-image-builder を使用してブート可能なコンテナーイメージを ISO イメージに変換した後、Anaconda とキックスタートを使用して ISO イメージをデプロイし、コンテナーイメージをインストールできます。インストール可能なブート ISO には、設定済みの **ostreecontainer** キックスタートファイルがすでに含まれています。このファイルは、カスタムコンテナーイメージのプロビジョニングに使用できます。

警告

rpm-ostree を使用して変更を加えたり、コンテンツをインストールしたりすることはサポートされていません。

前提条件

- Red Hat から、アーキテクチャー用の 9.4 Boot ISO をダウンロードした。RH ブートイメージのダウンロード を参照してください。

手順

1. **ostreecontainer** キックスタートファイルを作成します。以下に例を示します。

```
# Basic setup
text
network --bootproto=dhcp --device=link --activate
# Basic partitioning
clearpart --all --initlabel --disklabel=gpt
reqpart --add-boot
part / --grow --fstype xfs

# Reference the container image to install - The kickstart
# has no %packages section. A container image is being installed.
ostreecontainer --url registry.redhat.io/rhel9/bootc-image-builder:latest

firewall --disabled
services --enabled=sshd

# Only inject a SSH key for root
rootpw --iscrypted locked
sshkey --username root "<your key here>"
reboot
```

2. 9.4 Boot ISO インストールメディアを使用してシステムを起動します。
   a. カーネル引数に、次の内容のキックスタートファイルを追加します。

   ```
   inst.ks=http://<path_to_your_kickstart>
   ```

3. **CTRL+X** を押してシステムを起動します。

次のステップ

- コンテナーイメージをデプロイした後、イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

関連情報

- ostreecontainer ドキュメント
- bootc upgrade fails when using local rpm-ostree modifications solution

4.4. カスタム ISO コンテナーイメージのデプロイ

bootc-image-builder を使用して、ブート可能なコンテナーイメージを ISO イメージに変換します。これにより、コンテナーイメージのコンテンツが ISO ディスクイメージに埋め込まれることを除き、ダウンロード可能な RHEL ISO に似たシステムが作成されます。インストール中にネットワークにアクセスする必要はありません。次に、**bootc-image-builder** から作成した ISO ディスクイメージを、ベアメタルシステムにインストールします。

前提条件

- カスタマイズされたコンテナーイメージを作成した。

手順

1. **bootc-image-builder** を使用して、カスタムインストーラー ISO ディスクイメージを作成します。bootc-image-builder を使用した ISO イメージの作成 を参照してください。
2. ISO ディスクイメージを USB フラッシュドライブにコピーします。
3. USB スティック内のコンテンツを使用して、非接続環境でベアメタルインストールを実行します。

次のステップ

- コンテナーイメージをデプロイした後、イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

4.5. PXE ブート経由での ISO のブート可能なコンテナーのデプロイ

ネットワークインストールを使用して、PXE ブート経由で RHEL ISO イメージをデプロイし、ISO のブート可能なコンテナーイメージを実行できます。

前提条件

- Red Hat から、アーキテクチャー用の 9.4 Boot ISO をダウンロードした。RH ブートイメージのダウンロード を参照してください。
- サーバーが PXE ブート用に設定されている。以下のいずれかのオプションを選択します。
  - HTTP クライアントの場合は、HTTP ブートおよび PXE ブート用の DHCPv4 サーバーの設定 を参照してください。
  - UEFI ベースのクライアントの場合は、UEFI ベースのクライアント向けに TFTP サーバーを設定する を参照してください。
  - BIOS ベースのクライアントの場合は、BIOS ベースのクライアント向けに TFTP サーバーを設定する を参照してください。
- クライアント (ISO イメージをインストールするシステムとも呼ばれる) がある。

手順

1. RHEL インストール ISO イメージを、HTTP サーバーにエクスポートします。これにより、PXE ブートサーバーでは、PXE クライアントにサービスを提供する準備が整いました。
2. クライアントを起動して、インストールを開始します。
3. ブートソースを指定するよう求められたら、PXE ブートを選択します。ブートオプションが表示されない場合は、キーボードの Enter キーを押すか、起動画面が開くまで待ちます。
4. Red Hat Enterprise Linux の起動画面で、必要なブートオプションを選択し、Enter キーを押します。
5. ネットワークインストールを開始します。

次のステップ

- イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

関連情報

- PXE を使用してネットワークからインストールするための準備
- PXE を使用したネットワークからのインストールの起動

4.6. BOOTC-IMAGE-BUILDER を使用したディスクイメージのビルド、設定、起動

Containerfile を使用して、カスタムイメージに設定を注入できます。

手順

1. ディスクイメージを作成します。次の例は、ディスクイメージにユーザーを追加する方法を示します。

```
[[blueprint.customizations.user]]
name = "user"
password = "pass"
key = "ssh-rsa AAA ... user@email.com"
groups = ["wheel"]
```

   - **name** - ユーザー名。必須です。
   - **password** - 暗号化されていないパスワード。必須ではありません。
   - **key** - 公開 SSH キーの内容。必須ではありません。
   - **groups** - ユーザーを追加するグループの配列。必須ではありません。

2. **bootc-image-builder** を実行し、次の引数を渡します。

```
$ sudo podman run \
    --rm \
    -it \
    --privileged \
    --pull=newer \
    --security-opt label=type:unconfined_t \
    -v $(pwd)/config.toml:/config.toml \
    -v $(pwd)/output:/output \
    registry.redhat.io/rhel9/bootc-image-builder:latest \
    --type qcow2 \
    --config config.toml \
    quay.io/<namespace>/<image>:<tag>
```

3. たとえば、**virt-install** を使用して仮想マシンを起動します。

```
$ sudo virt-install \
  --name bootc \
  --memory 4096 \
  --vcpus 2 \
  --disk qcow2/disk.qcow2 \
  --import \
  --os-variant rhel9
```

検証

- SSH でシステムにアクセスします。

```
# ssh -i /<path_to_private_ssh-key> <user1>@<ip-address>
```

次のステップ

- コンテナーイメージをデプロイした後、イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

4.7. BOOTC を使用したコンテナーイメージのデプロイ

bootc を使用すると、信頼できるソースとなるコンテナーが得られます。基本的なビルドインストーラーが含まれており、**bootc install to-disk** または **bootc install to-filesystem** として使用できます。コンテナーイメージに基本的なインストーラーが含まれるため、**bootc install** の方法を使用すれば、コンテナーイメージをデプロイするために追加の手順を実行する必要はありません。

Image Mode for RHEL では、デフォルトのパスワードや SSH キーを持たないイメージなど、未設定のイメージをインストールできます。

RHEL ISO イメージを使用して、デバイスへのベアメタルインストールを実行します。

前提条件

- Red Hat から、アーキテクチャー用の 9.4 Boot ISO をダウンロードした。RH ブートイメージのダウンロード を参照してください。
- 設定ファイルが作成されている。

手順

- 以下のように、実行中の ISO イメージに設定を注入します。

```
$ podman run --rm --privileged --pid=host -v /var/lib/containers:/var/lib/containers --
security-opt label=type:unconfined_t <image> bootc install to-disk <path-to-disk>
```

次のステップ

- コンテナーイメージをデプロイした後、イメージを更新し、変更をレジストリーにプッシュできます。RHEL のブート可能なイメージの管理 を参照してください。

4.8. TO-FILESYSTEM を使用した高度なインストール

bootc install には、**bootc install to-disk** と **bootc install to-filesystem** という 2 つのサブコマンドが含まれます。

- **bootc-install-to-filesystem** は、ターゲットファイルシステムへのインストールを実行します。
- **bootc install to-disk** サブコマンドは、独自の低レベルツールのセットで構成されます。これらのツールは独立して呼び出すこともできます。コマンドを構成するツールは次のとおりです。

a. **mkfs $fs /dev/disk**

- **mkfs.$fs /dev/disk**
- **mount /dev/disk /mnt**
- **bootc install to-filesystem --karg=root=UUID=<uuid of /mnt> --imgref $self /mnt**

4.8.1. bootc install to-existing-root の使用

bootc install to-existing-root は **install to-filesystem** のバリアントです。これを使用して、既存のシステムをターゲットコンテナーイメージに変換できます。

警告

この変換により、**/boot** および **/boot/efi** パーティションが削除され、既存の Linux インストールが削除される可能性があります。変換プロセスではファイルシステムが再利用されます。ユーザーデータは保持されますが、システムはパッケージモードで起動しなくなります。

前提条件

- 手順を完了するには、root 権限が必要です。
- ホスト環境とターゲットコンテナーのバージョンを一致させる必要があります。たとえば、ホストが RHEL 9 ホストの場合は、RHEL 9 コンテナーが必要です。RHEL カーネルとして **btrfs** を使用して Fedora ホストに RHEL コンテナーをインストールすると、そのファイルシステムはサポートされません。

手順

- 既存のシステムをターゲットコンテナーイメージに変換するには、次のコマンドを実行します。**-v/:/target** オプションを使用して、ターゲットの **rootfs** を渡します。

```
# podman run --rm --privileged -v /dev:/dev -v /var/lib/containers:/var/lib/containers -v
/:/target \
        --pid=host --security-opt label=type:unconfined_t \
        <image> \
        bootc install to-existing-root
```

このコマンドにより **/boot** 内のデータは削除されますが、既存のオペレーティングシステム内にある他のすべてのデータは自動的に削除されません。これにより、新しいイメージが以前のホストシステムからデータを自動的にインポートできるため便利です。したがって、コンテナーイメージ、データベース、ユーザーホームディレクトリーデータ、**/etc** 内の設定ファイルはすべて、その後の再起動後に **/sysroot** で使用できるようになります。

--root-ssh-authorized-keys /target/root/.ssh/authorized_keys を追加することで、**--root-ssh-authorized-keys** フラグを使用して root ユーザーの SSH キーを継承することもできます。以下に例を示します。

```
# podman run --rm --privileged -v /dev:/dev -v /var/lib/containers:/var/lib/containers -v
/:/target \
        --pid=host --security-opt label=type:unconfined_t \
```

```
        <image> \
        bootc install to-existing-root --root-ssh-authorized-keys
/target/root/.ssh/authorized_keys
```

第5章 RHEL のブート可能なイメージの管理

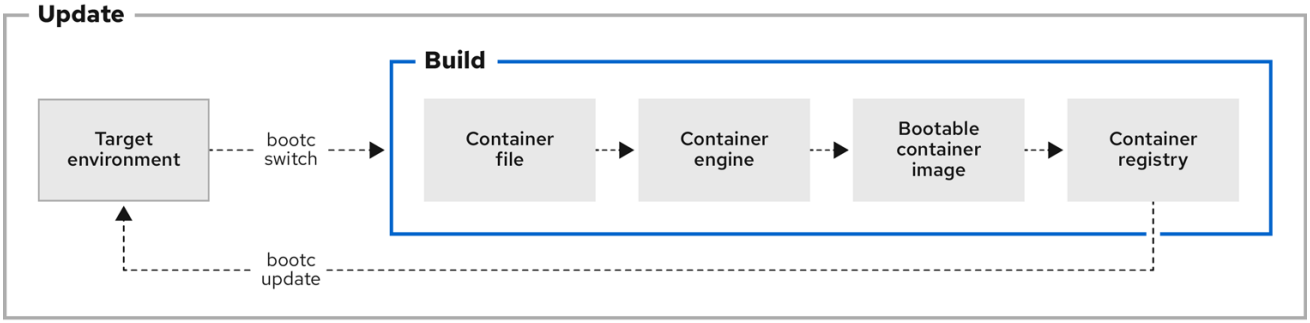
RHEL のブート可能なイメージをインストールおよびデプロイした後、システムの変更や更新などの管理操作をコンテナーイメージに対して実行できます。システムは、デプロイメント後にロールバックが可能なインプレーストランザクション更新をサポートします。

この種の管理は Day 2 管理ベースラインとも呼ばれ、コンテナーレジストリーから新しいオペレーティングシステムの更新をトランザクショナルに取得して、システムを起動します。同時に、障害発生時には手動または自動のロールバックをサポートします。

デフォルトで有効になっている自動更新を利用することもできます。**systemd service unit** と **systemd timer unit** ファイルがコンテナーレジストリーの更新をチェックし、システムに適用します。アプリケーションの更新など、さまざまなイベントで更新プロセスをトリガーできます。これらの更新を監視し、CI/CD パイプラインをトリガーする自動化ツールがあります。更新はトランザクションであるため、再起動が必要です。より高度なロールアウトやスケジュールされたロールアウトが必要な環境では、自動更新を無効にし、**bootc** ユーティリティーを使用してオペレーティングシステムを更新する必要があります。

詳細は、Day 2 operations support を参照してください。

図5.1 インストールされたオペレーティングシステムを手動で更新し、必要に応じて変更をコンテナーイメージの参照を変更したり、ロールバックしたりする



5.1. コンテナーイメージ参照の切り替え

bootc switch コマンドを使用して、アップグレードに使用するコンテナーイメージ参照を変更できます。たとえば、ステージタグから実稼働タグに切り替えることができます。**bootc switch** コマンドは、**bootc upgrade** コマンドと同じ操作を実行し、さらにコンテナーイメージ参照を変更します。

既存の **ostree-based** コンテナーイメージ参照を手動で切り替えるには、**bootc switch** コマンドを使用します。

警告

rpm-ostree を使用して変更を加えたり、コンテンツをインストールしたりすることはサポートされていません。

前提条件

- **bootc** を使用して起動したシステム。

手順

- 以下のコマンドを実行します。

```
$ bootc switch [--apply] quay.io/<namespace>/<image>:<tag>
```

システム変更時に再起動などのアクションを自動的に実行する場合は、必要に応じて **--apply** オプションを使用できます。

注記

bootc switch コマンドは **bootc upgrade** と同じ効果があります。唯一の違いは、コンテナーイメージ参照が変更されることです。これにより、ホスト SSH キーやホームディレクトリーなど、**/etc** および **/var** 内の既存の状態を保持できます。

関連情報

- **bootc-switch** の man ページ

5.2. BOOTC イメージ INITRAMFS へのモジュールの追加

rhel9/rhel-bootc イメージは、**dracut** インフラストラクチャーを使用して、イメージのビルド中に初期 RAM ディスク (**initrd**) を構築します。**initrd** は、コンテナー内の **/usr/lib/modules/$kver/initramfs.img** の場所に構築および追加されます。

ドロップイン設定ファイルを使用すると、**dracut** の設定をオーバーライドできます。ファイルは **/usr/lib/dracut/dracut.conf.d/<50-custom-added-modules.conf>** に配置できます。これにより、追加する必要があるモジュールを使用して **initrd** を再作成できます。

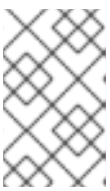
前提条件

- bootc を使用して起動したシステム。

手順

- コンテナービルドの一部として **initrd** を再作成します。

```
FROM <baseimage>
COPY <50-custom-added-modules>.conf /usr/lib/dracut/dracut.conf.d
RUN set -x; kver=$(cd /usr/lib/modules && echo *); dracut -vf
/usr/lib/modules/$kver/initramfs.img $kver
```



注記

デフォルトでは、このコマンドは実行中のカーネルバージョンをプルしようとするため、エラーが発生します。エラーを回避するために、ターゲットのカーネルバージョンを **dracut** に明示的に渡してください。

5.3. INITRD の変更と再生成

デフォルトのコンテナーイメージには、**/usr/lib/modules/$kver/initramfs.img** に事前に生成された初期 RAM ディスク (initrd) が含まれています。たとえば、dracut モジュールを追加するために **initrd** を再生成するには、次の手順に従います。

手順

1. ドロップイン設定ファイルを作成します。以下に例を示します。

```
dracutmodules = "module"
```

2. ドロップイン設定ファイルを、**dracut** が通常使用する場所 (**/usr**) に配置します。以下に例を示します。

```
/usr/lib/dracut/dracut.conf.d/50-custom-added-modules.conf
```

3. コンテナービルドの一部として **initrd** を再生成します。ターゲットのカーネルバージョンを **dracut** に明示的に渡す必要があります。dracut が実行中のカーネルバージョンをプルしようとしたときに、エラーが発生する可能性があるためです。以下に例を示します。

```
FROM <baseimage>
COPY 50-custom-added-modules.conf /usr/lib/dracut/dracut.conf.d
RUN set -x; kver=$(cd /usr/lib/modules && echo *); dracut -vf
/usr/lib/modules/$kver/initramfs.img $kver
```

5.4. インストールされたオペレーティングシステムからの更新の手動実行

Image Mode for RHEL のインストールは 1 回限りのタスクです。システムの変更や更新などのその他の管理タスクは、変更をコンテナーレジストリーにプッシュすることで実行できます。

Image Mode for RHEL を使用する場合、システムを手動で更新することを選択できます。手動更新は、Ansible などを使用して更新を自動的に実行できる場合でも便利です。自動更新はデフォルトで有効になっているため、手動更新を実行するには自動更新をオフにする必要があります。これは、次のいずれかの方法を選択して実行できます。

- **bootc upgrade** コマンドの実行
- **systemd** タイマーファイルの変更

5.5. 自動更新の無効化

手動更新を実行するには、自動更新をオフにする必要があります。これは、以下の手順に記載のいずれかの方法を選択して実行できます。

手順

- コンテナービルドのタイマーを無効にします。
  - **bootc upgrade** コマンドを実行します。

```
$ systemctl mask bootc-fetch-apply-updates.timer
```

  - **systemd** タイマーファイルを変更します。**systemd** の "ドロップイン" を使用して、タイマーをオーバーライドします。次の例では、更新は週に 1 回スケジュールされています。

1. 次の内容で **updates.conf** ファイルを作成します。

```
[Timer]
# Clear previous timers
OnBootSec= OnBootSec=1w OnUnitInactiveSec=1w
```

2. 作成したファイルにコンテナーを追加します。

```
$ mkdir -p /usr/lib/systemd/system/bootc-fetch-apply-updates.timer.d
$ cp updates.conf /usr/lib/systemd/system/bootc-fetch-apply-updates.timer.d
```

5.6. インストールされたオペレーティングシステムの手動更新

レジストリーから手動で更新を取得し、新しい更新でシステムを起動するには、**bootc upgrade** を使用します。このコマンドは、インストールされたオペレーティングシステムからコンテナーイメージレジストリーへの、トランザクショナルなインプレース更新を取得します。このコマンドはレジストリーをクエリーし、次回の起動のために更新されたコンテナーイメージをキューに入れます。デフォルトでは実行中のシステムを変更せずに、ベースイメージへの変更をステージングします。

手順

- 以下のコマンドを実行します。

```
$ bootc upgrade [--apply]
```

apply 引数はオプションです。システム変更時に再起動などのアクションを自動的に実行する場合は、この引数を使用できます。

注記

bootc upgrade および **bootc update** コマンドはエイリアスです。

関連情報

- **bootc-upgrade** の man ページ

5.7. 更新されたオペレーティングシステムからのロールバックの実行

bootc rollback コマンドを使用すると、以前のブートエントリーにロールバックして変更を元に戻すことができます。このコマンドは、**rollback** 対象のデプロイメントを次回の起動のキューに追加することで、ブートローダーエントリーの順序を変更します。現在のデプロイメントはロールバックになります。キュー内の適用されなかったアップグレードなどのステージングされた変更は、すべて破棄されます。

ロールバックが完了すると、システムが再起動し、更新タイマーが 1 - 3 時間以内に実行し、ロールバックしたイメージにシステムが自動的に更新されて再起動されます。

警告

ロールバックを実行すると、自動更新をオフにしない限り、システムは自動的に再度更新されます。自動更新の無効化 を参照してください。

前提条件

- システムの更新を実行した。

手順

- 以下のコマンドを実行します。

```
$ bootc rollback [-h|--help] [-V|--version]
```

注記

bootc rollback コマンドは **bootc upgrade** と同じ効果があります。唯一の違いは、コンテナーイメージが追跡されることです。これにより、ホスト SSH キーやホームディレクトリーなど、**/etc** および **/var** 内の既存の状態を保持できます。

検証

- **systemd journal** を使用して、検出されたロールバック呼び出しに関するログに記録されたメッセージを確認します。

```
$ journalctl -b
```

次のようなログが表示されます。

```
MESSAGE_ID=26f3b1eb24464d12aa5e7b544a6b5468
```

関連情報

- **bootc-rollback** の man ページ

5.8. システムグループへの更新のデプロイ

Containerfile を変更することで、オペレーティングシステムの設定を変更できます。その後、コンテナーイメージをビルドして、レジストリーにプッシュできます。次回のオペレーティングシステムの起動時に、更新が適用されます。

bootc switch コマンドを使用して、コンテナーイメージソースを変更することもできます。コンテナーレジストリーは信頼できるソースです。コンテナーイメージ参照の切り替え を参照してください。

通常、システムグループに更新をデプロイする場合、セントラル管理サービスを使用できます。これにより、各システムにインストールされ、セントラルサービスに接続するクライアントを提供できます。多くの場合、管理サービスでは、クライアントが 1 回限りの登録を実行する必要があります。以下は、

システムグループに更新をデプロイする方法の例です。必要に応じて、これを変更して永続的な **systemd** サービスを作成できます。

注記

理解しやすいように、以下の例の Containerfile は最適化されていません。たとえば、イメージ内に複数のレイヤーが作成されないように最適化するには、RUN を 1 回呼び出します。

RHEL イメージのイメージモードにクライアントをインストールし、起動時に実行してシステムを登録できます。

前提条件

- 管理クライアントが、**cron** ジョブまたは別の **systemd** サービスを使用して、今後のサーバーへの接続を処理する。

手順

- 次の特徴を持つ管理サービスを作成します。管理サービスは、システムをいつアップグレードするかを決定します。

  1. ベースイメージに **bootc-fetch-apply-updates.timer** が含まれている場合は無効にします。
  2. **dnf** を使用するか、クライアントに適用される他の方法を使用してクライアントをインストールします。
  3. 管理サービスの認証情報をイメージに注入します。

5.9. インベントリーの健全性の確認

ヘルスチェックは Day 2 操作の 1 つです。コンテナーイメージとコンテナー内で実行中のイベントのシステム健全性を、手動で確認できます。

コマンドラインでコンテナーを作成することにより、ヘルスチェックを設定できます。**podman inspect** または **podman ps** コマンドを使用して、コンテナーのヘルスチェックのステータスを表示できます。

podman events コマンドを使用して、Podman で発生するイベントを監視および出力できます。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。

ヘルスチェックとイベントの詳細は、コンテナーの監視 の章を参照してください。

5.10. 自動化と GITOPS

アプリケーションの更新などのイベントにより更新プロセスをトリガーできるように、CI/CD パイプラインを使用してビルドプロセスを自動化できます。これらの更新を追跡し、CI/CD パイプラインをトリガーする自動化ツールを使用できます。パイプラインは、トランザクショナルなバックグラウンドのオペレーティングシステム更新を使用して、システムを最新の状態に保ちます。

第6章 IMAGE MODE FOR RHEL でのファイルシステムの管理

現在、Image Mode for RHEL では、バックエンドとして OSTree が使用され、ストレージ用に **composefs** がデフォルトで有効になっています。**/opt** および **/usr/local** パスはプレーンディレクトリーであり、**/var** へのシンボリックリンクではありません。これにより、たとえば **/opt** に書き込む派生コンテナーイメージにサードパーティーのコンテンツを簡単にインストールできるようになります。

6.1. /SYSROOT による物理ルートと論理ルート

システムが完全に起動すると、**chroot** に似た状態になり、オペレーティングシステムは現在実行中のプロセスとその子プロセスの見かけ上のルートディレクトリーを変更します。物理ホストのルートファイルシステムは **/sysroot** にマウントされます。**chroot** ファイルシステムはデプロイメントルートと呼ばれます。

残りのファイルシステムパスは、システムブートの最終ターゲットとして使用されるデプロイメントルートの一部です。システムは、**ostree=kernel** 引数を使用してデプロイメントルートを検索します。

/usr

このファイルシステムは、すべてのオペレーティングシステムのコンテンツを **/usr** に保存し、**/bin** などのディレクトリーは **/usr/bin** へのシンボリックリンクとして機能します。

注記

composefs が有効になっている **/usr** は / と変わりません。両方のディレクトリーは同じ不変イメージの一部であるため、bootc システムで完全な **UsrMove** を実行する必要はありません。

/usr/local

ベースイメージは、デフォルトのディレクトリーとして **/usr/local** に設定されています。

/etc

/etc ディレクトリーにはデフォルトで変更可能な永続状態が含まれますが、**etc.transient config** オプションを有効にすることがサポートされています。ディレクトリーが変更可能な永続状態にある場合は、アップグレード全体で 3 方向のマージが実行します。

- 新しいデフォルトの **/etc** をベースとして使用します
- 現在の **/etc** と以前の **/etc** の差分を新しい **/etc** ディレクトリーに適用します。
- 同じデプロイメントのデフォルトの **/usr/etc** とは異なる、ローカルで変更されたファイルを **/etc** に保持します。

ostree-finalize-staged.service は、新しいブートローダーエントリーを作成する前に、シャットダウン時にこれらのタスクを実行します。

これは、Linux システムの多くのコンポーネントが **/etc** ディレクトリーにデフォルトの設定ファイルを置いて出荷されるために発生します。デフォルトパッケージに含まれていない場合でも、デフォルトではソフトウェアは **/etc** 内の設定ファイルのみをチェックします。**/etc** の明確なバージョンを持たない、bootc イメージベースではない更新システムは、インストール時にのみ設定され、インストール後は変更されません。これにより、**/etc** システムの状態が初期イメージバージョンの影響を受けるようになり、たとえば **/etc/sudoers.conf** に変更を適用する際に問題が発生し、外部からの介入が必要になる可能性があります。ファイル設定の詳細は、RHEL のブート可能なコンテナーイメージのビルドおよびテスト を参照してください。

/var

/var ディレクトリーの内容はデフォルトで永続的です。ネットワークでも **tmpfs** でも、**/var** またはサブディレクトリーのマウントポイントを永続的にすることもできます。

/var ディレクトリーは1つだけです。別個のパーティションでない場合、物理的には **/var** ディレクトリーは **/ostree/deploy/$stateroot/var** へのバインドマウントとなり、利用可能なブートローダーエントリーのデプロイメント間で共有されます。

デフォルトでは、**/var** 内のコンテンツはボリュームとして機能します。つまり、コンテナーイメージのコンテンツは最初のインストール時にコピーされ、その後は更新されません。

/var ディレクトリーと **/etc** ディレクトリーは異なります。比較的小さな設定ファイルには **/etc** を使用でき、必要な設定ファイルは、多くの場合 **/usr** 内のオペレーティングシステムバイナリーにバインドされます。**/var** ディレクトリーには、システムログ、データベースなどの任意の大きさのデータが含まれており、デフォルトでは、オペレーティングシステムの状態がロールバックされてもロールバックされません。

たとえば、**dnf downgrade postgresql** などの更新を実行しても、**/var/lib/postgres** 内の物理データベースには影響しません。同様に、**bootc update** または **bootc rollback** を実行しても、このアプリケーションデータには影響しません。

/var を別々にしておくと、新しいオペレーティングシステムの更新を適用する前にステージングをスムーズに実行できるようになります。つまり、更新はダウンロードされて準備完了ですが、再起動時にのみ有効になります。同じことが Docker ボリュームにも当てはまり、アプリケーションコードとそのデータが切り離されます。

アプリケーションに **/var/lib/postgresql** などの事前に作成されたディレクトリー構造を持たせたい場合に、このケースを使用できます。これには **systemd tmpfiles.d** を使用します。ユニットで **StateDirectory=<directory>** を使用することもできます。

その他のディレクトリー

コンテナーイメージ内の **/run**、**/proc**、またはその他の API ファイルシステム内のコンテンツを出荷することはサポートされていません。それ以外にも、**/usr** や **/opt** などの他のトップレベルディレクトリーは、コンテナーイメージとともにライフサイクル化されます。

/opt

composefs を使用する **bootc** では、**/opt** ディレクトリーは、**/usr** などの他の最上位ディレクトリーと同様に読み取り専用になります。

ソフトウェアが **/opt/exampleapp** 内の独自のディレクトリーに書き込む必要がある場合は、ログファイルなどの操作のために、たとえば **/var** にリダイレクトするシンボリックリンクを使用するのが一般的なパターンです。

```
RUN rmdir /opt/exampleapp/logs && ln -sr /var/log/exampleapp /opt/exampleapp/logs
```

オプションで、これらのマウントを動的に実行するサービスを起動するように systemd ユニットを設定することもできます。以下に例を示します。

```
BindPaths=/var/log/exampleapp:/opt/exampleapp/logs
```

一時ルートを有効にする

デフォルトで完全に一時的な書き込み可能な **rootfs** を有効にするには、**prepare-root.conf** で次のオプションを設定します。

```
[root]
transient = true
```

これにより、ソフトウェアは、永続化する必要のあるコンテンツに対して **/var** へのシンボリックリンクを使用して、一時的に **/opt** に書き込むことができます。

6.2. バージョンの選択と起動

Image Mode for RHEL では、**s390x** アーキテクチャーを除き、デフォルトで GRUB が使用されます。現在システムで使用可能な Image Mode for RHEL の各バージョンには、メニューエントリーがあります。

メニューエントリーは、Linux カーネル、**initramfs**、および OSTree コミットにリンクするハッシュで構成される OSTree デプロイメントを参照します。これは、**ostree=kernel** 引数を使用して渡すことができます。

起動時に、OSTree はカーネル引数を読み取り、ルートファイルシステムとして使用するデプロイメントを決定します。パッケージのインストール、カーネル引数の追加など、システムに対する更新または変更ごとに、新しいデプロイメントが作成されます。

これにより、更新によって問題が発生した場合に、以前のデプロイメントにロールバックできるようになります。

第7章 付録: IMAGE MODE FOR RHEL でのユーザー、グループ、SSH キー、シークレットの管理

Image Mode for RHEL でのユーザー、グループ、SSH キー、およびシークレットの管理について説明します。

7.1. ユーザーとグループの設定

Image Mode for RHEL は、汎用的なオペレーティングシステムの更新および設定メカニズムです。ユーザーまたはグループを設定するために使用することはできません。唯一の例外は、**--root-ssh-authorized-keys** オプションを持つ **bootc install** コマンドです。

汎用ベースイメージのユーザーとグループの設定

通常、ディストリビューションのベースイメージには設定がありません。セキュリティーリスクがあるため、汎用イメージ内の公開されている秘密鍵を使用してパスワードと SSH キーを暗号化しないでください。

systemd 認証情報を使用した SSH キーの注入

一部の環境では、**systemd** を使用して、root パスワードまたは SSH **authorized_keys** ファイルを注入できます。たとえば、System Management BIOS (SMBIOS) を使用して、SSH キーシステムファームウェアを注入します。これは、**qemu** などのローカルな仮想化環境で設定できます。

cloud-init を使用したユーザーと SSH キーの注入

多くの Infrastructure as a Service (IaaS) および仮想化システムは、**cloud-init** や **ignition** などのソフトウェアによって一般的に処理されるメタデータサーバーを使用します。AWS instance metadata を参照してください。**cloud-init** または Ignition は、使用しているベースイメージに含まれる場合があります。または、独自の派生イメージにインストールすることもできます。このモデルでは、SSH 設定はブート可能なイメージの外部で管理されます。

コンテナーまたはユニットのカスタムロジックを使用したユーザーと認証情報の追加

cloud-init などのシステムには特権がありません。コンテナーイメージを起動する方法で、認証情報を管理するための任意のロジックを注入できます。たとえば、**systemd** ユニットを使用できます。認証情報を管理するには、FreeIPA などのカスタムのネットワークホストソースを使用できます。

コンテナービルドでのユーザーと認証情報の静的な追加

パッケージ指向のシステムでは、次のコマンドにより、派生ビルドを使用してユーザーと認証情報を注入できます。

```
RUN useradd someuser
```

shadow-utils のデフォルトの **useradd** 実装に問題がある場合があります。ユーザーとグループの ID が動的に割り当てられることにより、ドリフトが発生する可能性があります。

ユーザーとグループのホームディレクトリーと /var ディレクトリー

永続的に **/home → /var/home** に設定されたシステムの場合、最初のインストール後にコンテナーイメージで行われた **/var** への変更は、その後の更新には適用されません。
たとえば、コンテナービルドに **/var/home/someuser/.ssh/authorized_keys** を注入した場合、既存のシステムは更新された **authorized_keys** ファイルを取得しません。

systemd ユニットでの DynamicUser=yes の使用

システムユーザーの場合、可能な場合は **systemd DynamicUser=yes** オプションを使用します。これは、潜在的な UID または GID のドリフトを回避できるため、パッケージのインストール時にユーザーまたはグループを割り当てるパターンよりもはるかに優れています。

systemd-sysusers の使用

たとえば、派生ビルドでは **systemd-sysusers** を使用します。詳細は、**systemd-sysusers** のドキュメントを参照してください。

```
COPY mycustom-user.conf /usr/lib/sysusers.d
```

sysusers ツールは、必要に応じて起動時に従来の **/etc/passwd** ファイルに変更を加えます。**/etc** が永続的であれば、**UID** または **GID** のドリフトを回避できます。つまり、**UID** または **GID** の割り当ては、特定のマシンがどのようにアップグレードされてきたかによって異なります。

systemd JSON ユーザーレコードの使用

JSON user records の **systemd** に関するドキュメントを参照してください。**sysusers** とは異なり、これらのユーザーの標準的な状態は **/usr** にあります。後続のイメージでユーザーレコードが破棄されると、そのレコードはシステムからも消えます。

nss-altfiles の使用

nss-altfiles を使用すると、**systemd** の JSON ユーザーレコードを削除できます。これは、システムユーザーを **/usr/lib/passwd** と **/usr/lib/group** に分割します。OSTree プロジェクトが **/etc/passwd** に関連する **/etc** の 3 方向マージを処理する方法と整合します。現在、ローカルシステムで **/etc/passwd** ファイルが何らかの方法で変更された場合、その後コンテナーイメージの **/etc/passwd** に対して行われた変更は適用されません。
rpm-ostree によってビルドされたベースイメージでは、**nns-altfiles** がデフォルトで有効になっています。

また、ベースイメージには、UID または GID のドリフトを回避するために、NSS ファイルによって事前に割り当てられ、管理されるシステムユーザーがあります。

派生コンテナービルドでは、たとえば **/usr/lib/passwd** にユーザーを追加することもできます。**sysusers.d** または **DynamicUser=yes** を使用します。

ユーザーのマシンローカル状態

ファイルシステムのレイアウトは、ベースイメージによって異なります。
デフォルトでは、ユーザーデータはベースイメージに応じて、**/etc**、**/etc/passwd**、**/etc/shadow** および **groups** と、**/home** との両方に保存されます。ただし、いずれの汎用ベースイメージもマシンローカルな永続状態である必要があります。このモデルでは、**/home** は **/var/home/user** へのシンボリックリンクです。

システムプロビジョニング時のユーザーと SSH キーの注入

/etc と **/var** がデフォルトで永続化するように設定されたベースイメージの場合、Anaconda やキックスタートなどのインストーラーを使用してユーザーを注入できます。
通常、汎用インストーラーは 1 回限りのブートストラップ用に設計されています。その後、設定はミュータブルなマシンローカル状態になり、他のメカニズムを使用して Day 2 操作で変更できるようになります。

Anaconda インストーラーを使用して初期パスワードを設定できます。ただし、この初期パスワードを変更するには、**passwd** などの別のシステム内ツールが必要です。

これらのフローは **bootc-compatible** システムでも同様に機能します。異なるシステム内ツールに変更する必要なく、ユーザーが汎用ベースイメージを直接インストールすることをサポートします。

一時的なホームディレクトリー

多くのオペレーティングシステムのデプロイメントでは、永続的でミュータブルかつ実行可能な状態が最小限に抑えられます。これにより、ユーザーのホームディレクトリーが破損する可能性があります。
/home ディレクトリーを **tmpfs** として設定すると、再起動後にユーザーデータが確実に消去されます。このアプローチは、一時的な **/etc** ディレクトリーと組み合わせると特に効果的です。

たとえば、SSH **authorized_keys** やその他のファイルを注入するようにユーザーのホームディレクトリーをセットアップするには、**systemd tmpfiles.d** スニペットを使用します。

```
f~ /home/user/.ssh/authorized_keys 600 user user - <base64 encoded data>
```

SSH は、**/usr/lib/tmpfiles.d/<username-keys.conf** としてイメージ内に埋め込まれています。もう 1 つの例は、ネットワークからキーを取得して書き込むことができる、イメージに埋め込まれたサービスです。これは **cloud-init** で使用されるパターンです。

UID と GID のドリフト

/etc/passwd および同様のファイルは、名前と数値識別子間のマッピングです。マッピングが動的で、"ステートレス" なコンテナーイメージビルドと組み合わされると、問題が発生する可能性があります。各コンテナーイメージビルドでは、RPM のインストール順序やその他の理由により、UID が変更される場合があります。当該ユーザーが永続状態を維持する場合、これは問題になる可能性があります。このようなケースに対処するには、**sysusers.d** を使用するか、**DynamicUser=yes** を使用するように変換します。

7.2. IMAGE MODE FOR RHEL でのシークレットの注入

Image Mode for RHEL には、シークレットに関する独自のメカニズムがありません。いくつかのケースでは、以下のようにコンテナープルシークレットをシステムに注入できます。

- **bootc** が認証の必要なレジストリーから更新を取得するには、ファイルにプルシークレットを含める必要があります。次の例では、**creds** シークレットにレジストリープルシークレットが含まれています。

```
FROM registry.redhat.io/rhel9/bootc-image-builder:latest
COPY containers-auth.conf /usr/lib/tmpfiles.d/link-podman-credentials.conf
RUN --mount=type=secret,id=creds,required=true cp /run/secrets/creds /usr/lib/container-auth.json && \
    chmod 0600 /usr/lib/container-auth.json && \
    ln -sr /usr/lib/container-auth.json /etc/ostree/auth.json
```

  ビルドするには、**podman build --secret id=creds,src=$HOME/.docker/config.json** を実行します。コンテナーイメージに埋め込まれた共通の永続ファイル (例: **/usr/lib/container-auth.json**) へのシンボリックリンクを両方の場所に対して使用することで、**bootc** と Podman に単一のプルシークレットを使用します。

- Podman がコンテナーイメージを取得するには、**/etc/containers/auth.json** にプルシークレットを含めます。この設定では、2 つのスタックが **/usr/lib/container-auth.json** ファイルを共有します。

  コンテナービルドにシークレットを埋め込むことによるシークレットの注入

  レジストリーサーバーが適切に保護されている場合は、コンテナーイメージにシークレットを含めることができます。場合により、ブートストラップシークレットのみをコンテナーイメージに埋め込むことが実行可能なパターンになることがあります。特に、マシンをクラスターに対して認証するメカニズムと併用した場合が該当します。このパターンでは、プロビ

ジョニングツールは、ホストシステムの一部として実行されるか、コンテナーイメージの一部として実行されるかに関係なく、ブートストラップシークレットを使用して SSH キーや証明書などの他のシークレットを注入または更新します。

クラウドメタデータを使用したシークレットの注入

実稼働環境の Infrastructure as a Service (IaaS) システムのほとんどは、シークレット (特にブートストラップシークレット) をセキュアにホストできる、メタデータサーバーまたは同等のものをサポートします。コンテナーイメージには、これらのシークレットを取得するための **cloud-init** や **ignition** などのツールを含めることができます。

ディスクイメージにシークレットを埋め込むことによるシークレットの注入

bootstrap secrets は、ディスクイメージにのみ埋め込むことができます。たとえば、AMI や OpenStack などの入力コンテナーイメージからクラウドディスクイメージを生成する場合、ディスクイメージには、実質的にマシンローカル状態であるシークレットが含まれることがあります。これらをローテーションするには、追加の管理ツールを使用するか、ディスクイメージを更新する必要があります。

ベアメタルインストーラーを使用したシークレットの注入

インストーラーツールは通常、シークレットを使用した設定の注入をサポートします。

systemd 認証情報を使用したシークレットの注入

systemd プロジェクトには、認証情報のデータをセキュアに取得してシステムやサービスに渡すための認証情報の概念があります。これは、一部のデプロイメント方法に適用されます。詳細は、systemd credentials のドキュメントを参照してください。

関連情報

- Example bootable containers を参照してください。