Red Hat Enterprise Linux 9: Image Mode

Chapter 1: Introduction to image mode for Red Hat Enterprise Linux

Use image mode for RHEL to build, test, and deploy operating systems by using the same tools and techniques as application containers. Image mode for RHEL is available by using the registry.redhat.io/rhel9/rhel-bootc bootc image. The RHEL bootc images differ from the existing application Universal Base Images (UBI) in that they contain additional components necessary to boot that were traditionally excluded, such as, kernel, initrd, boot loader, firmware, among others.

⚠️ **Important**

Red Hat provides the `rhel9/rhel-9-bootc` container image as a Technology Preview. Technology Preview features provide early access to upcoming product innovations, enabling customers to test functionality and provide feedback during the development process. However, these features are not fully supported. Documentation for a Technology Preview feature might be incomplete or include only basic installation and configuration information. See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

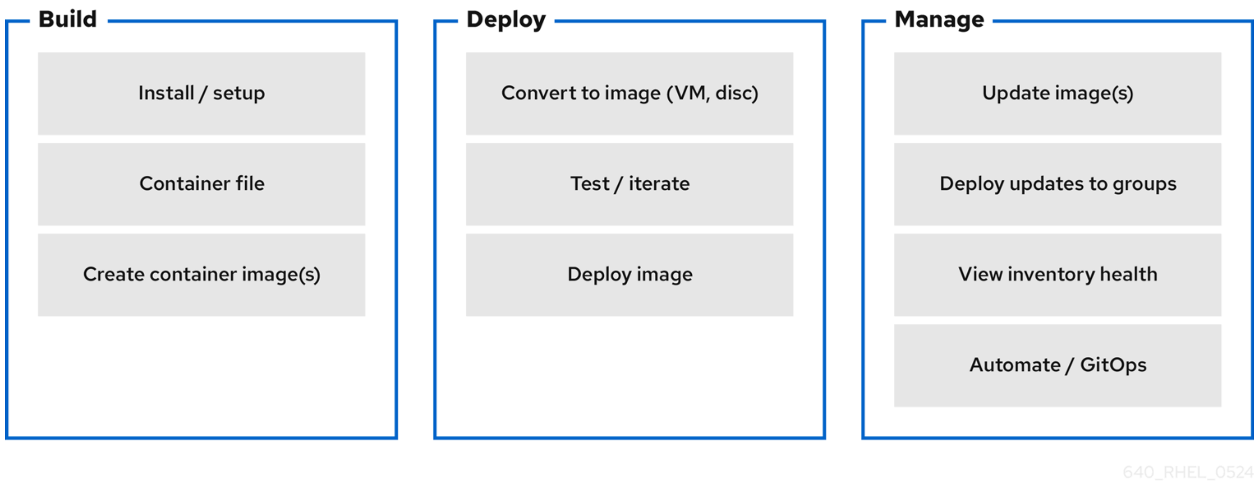

Figure 1.1. Building, deploying, and managing operating system by using image mode for RHEL

Red Hat provides bootc image for the following computer architectures:

- AMD and Intel 64-bit architectures (x86-64-v2)
- The 64-bit ARM architecture (ARMv8.0-A)

The benefits of image mode for RHEL occur across the lifecycle of a system. The following list contains some of the most important advantages:

Container images are easier to understand and use than other image formats and are fast to build

Containerfiles, also known as Dockerfiles, provide a straightforward approach to defining the content Containerfiles, also known as Dockerfiles, provide a straightforward approach to defining the content and build instructions for an image. Container images are often significantly faster to build and iterate on compared to other image creation tools.

Consolidate process, infrastructure, and release artifacts

As you distribute applications as containers, you can use the same infrastructure and processes to manage the underlying operating system.

Immutable updates

Just as containerized applications are updated in an immutable way, with image mode for RHEL, the operating system is also. You can boot into updates and roll back when needed in the same way that you use rpm-ostree systems.

> ⚠️**WARNING**
> The use of rpm-ostree to make changes, or install content, is not supported.

Portability across hybrid cloud environments

You can use bootc images across physical, virtualized, cloud, and edge environments. Although containers provide the foundation to build, transport, and run images, it is important to understand that after you deploy these bootc images, either by using an installation mechanism, or you convert them to a disk image, the system does not run as a container.

Bootc supports the following container image formats and disk image formats: OCI container format, ami, qcow2 (default), vmdk, anaconda-iso, raw, vhd and gce.

Containers help streamline the lifecycle of a RHEL system by offering the following possibilities:

Building container images

You can configure your operating system at a build time by modifying the Containerfile. Image mode for RHEL is available by using the `registry.redhat.io/rhel9/rhel-bootc` container image. You can use Podman, OpenShift Container Platform, or other standard container build tools to manage your containers and container images. You can automate the build process by using CI/CD pipelines.

Versioning, mirroring, and testing container images

You can version, mirror, introspect, and sign your derived bootc image by using any container tools such as Podman or OpenShift Container Platform.

Deploying container images to the target environment

You have several options on how to deploy your image:

- **Anaconda**: is the installation program used by RHEL. You can deploy all image types to the target environment by using Anaconda and Kickstart to automate the installation process.

- **bootc-image-builder**: is a containerized tool that converts the container image to different types of disk images, and optionally uploads them to an image registry or object storage.

- **bootc**: is a tool responsible for fetching container images from a container registry and installing them to a system, updating the operating system, or switching from an existing ostree-based system. The RHEL bootc image contains the bootc utility by default and works with all image types. However, remember that the rpm-ostree is not supported and must not be used to make changes.

Updating your operating system

The system supports in-place transactional updates with rollback after deployment. Automatic updates are on by default. A systemd service unit and systemd timer unit files check the container registry for updates and apply them to the system. As the updates are transactional, a reboot is required. For environments that require more sophisticated or scheduled rollouts, disable auto updates and use the bootc utility to update your operating system.

RHEL has two deployment modes. Both provide the same stability, reliability, and performance during deployment. See their differences:

1. **Package mode**: You can build package-based images and OSTree images by using RHEL image builder, and you can manage the package mode images by using composer-cli or web console. The operating system uses RPM packages and is updated by using the dnf package manager. The root filesystem is mutable. However, the operating system cannot be managed as a containerized application. See Composing a customized RHEL system image product documentation.

2. **Image mode**: a container-native approach to build, deploy, and manage RHEL. The same RPM packages are delivered as a base image and updates are deployed as a container image. The root filesystem is immutable by default, except for /etc and /var, with most content coming from the container image.

You can choose to use either the Image mode or the Package mode deployment to build, test, and share your operating system. Image mode additionally enables you to manage your operating system in the same way as any other containerized application.

Chapter 2: Building and Testing RHEL bootc images

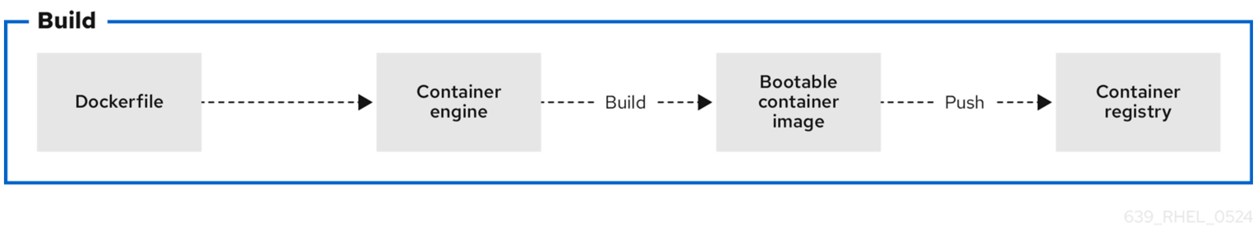

Figure 2.1. Building an image by using instructions from a Containerfile, testing the container, pushing an image to a registry, and sharing it with others

A general Containerfile structure is the following:

```
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]
```

The available commands that are usable inside a `Containerfile` and a `Dockerfile` are equivalent.

However, the following commands in a `Containerfile` are ignored when the `rhel-9-bootc` image is installed to a system:

- `ENTRYPOINT` and `CMD` (OCI: `Entrypoint/Cmd`): you can set `CMD /sbin/init` instead.
- `ENV` (OCI: `Env`): change the `systemd` configuration to configure the global system environment.
- `EXPOSE` (OCI: `exposedPorts`): it is independent of how the system firewall and network function at runtime.
- `USER` (OCI: `User`): configure individual services inside the RHEL bootc to run as unprivileged users instead.

The `rhel-9-bootc` container image reuses the OCI image format.

- The `rhel-9-bootc` container image ignores the container config section (`Config`) when it is installed to a system.
- The `rhel-9-bootc` container image does not ignore the container config section (`Config`) when you run this image by using container runtimes such as `podman` or `docker`

Note: Building custom `rhel-bootc` base images is not supported in this release

Lab 1: Build a container image

1. Create a `Containerfile` and index.html in the webvm directory

```
$ cd
$ mkdir webvm; cd webvm
$ echo Hello World > index.html
$ cat Containerfile
FROM registry.redhat.io/rhel9/rhel-bootc
ADD /etc/yum.repos.d/rhel_dvd.repo /etc/yum.repos.d/
RUN dnf -y install cloud-init && \
    systemctl enable httpd && \
    dnf clean all
ADD index.html /var/www/html
```

   This Containerfile example adds the cloud-init tool, so it automatically fetches SSH keys and can run scripts from the infrastructure and also gather configuration and secrets from the instance metadata. For example, you can use this container image for pre-generated AWS or KVM guest systems.
   You can use the following Containerfile if you don't have access to registry.redhat.io.

```
FROM quay.io/centos-bootc/centos-bootc-dev:stream9
RUN dnf -y install httpd && \
    systemctl enable httpd && \
    dnf clean all
ADD index.html /var/www/html
```

2. Build the *`<image>`* image by using `Containerfile` in the current directory:

```
$ podman build -t quay.io/<namespace>/webvm:1.0 .
```

3. List all images:

```
$ podman images
REPOSITORY                     TAG      IMAGE ID       CREATED              SIZE
quay.io/<namespace>/webvm      1.0      b28cd00741b3   About a minute ago   2.1 GB
```

4. Push image to registry:

```
$ podman login quay.io
  Username: <enter your quay.io account>
  Password: <enter your password>

$ podman push quay.io/<namespace>/webvm:1.0
```

5. Login to quay.io in your browser using your Red Hat Login ID. Add a ***latest*** tag for the image webvm:1.0.

NOTE: If you are using registry.redhat.io/rhel9/rhel-bootc image, you need to login to the private registry using your Red Hat ID with the command podman login `registry.redhat.io`

Chapter 3: Creating bootc compatible base disk images with bootc-image-builder

With the `bootc-image-builder` tool, you can convert bootc images into disk images for a variety of different platforms and formats. Converting bootc images into disk images is equivalent to installing a bootc. After you deploy these disk images to the target environment, you can update them directly from the container registry.

📋**Note**
The `bootc-image-builder` can only pull and use images from public container repositories. Building base disk images which come from private registries by using `bootc-image-builder` is not supported in this release. If your container image is stored in a private repository, `bootc-image-builder` cannot pull the image because it is not able to authenticate to the registry. If you need to use an image from a private repository, you must authenticate to the registry first and then pull the container image before you use it with `bootc-image-builder`. After pulling the image, you can run the bootc-image-builder command using the `--local` option.

The `bootc-image-builder tool` supports generating the following image types:

- Disk image formats, such as ISO, suitable for disconnected installations.

- Virtual disk images formats, such as:
  - QEMU copy-on-write (QCOW2)

  - Amazon Machine Image (AMI)/ — Raw
  - Virtual Machine Image (VMI)

Deploying from a container image is beneficial when you run VMs or servers because you can achieve the same installation result. That consistency extends across multiple different image types and platforms when you build them from the same container image. Consequently, you can minimize the effort in maintaining operating system images across platforms. You can also update systems that you deploy from these disk images by using the `bootc` tool, instead of re-creating and uploading new disk images with `bootc-image-builder`.

📋**Note**
Generic base container images do not include any default passwords or SSH keys. Also, the disk images that you create by using the `bootc-image-builder` tool do not contain the tools that are available in common disk images, such as `cloud-init`. These disk images are transformed container images only.

Although you can deploy a `rhel-9-bootc` image directly, you can also create your own customized images that are derived from this bootc image. The `bootc-image-builder` tool takes the `rhel-9-bootc` OCI container image as an input.

Lab 2: Installing bootc-image-builder and creating QCOW2 images

1. Login to `registry.redhat.io`. Make sure to use sudo. The subsequent steps need to be executed in privileged mode.

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

2. Install the `bootc-image-builder` tool:

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

3. List all images:

```
$ sudo podman images
REPOSITORY                                    TAG       IMAGE ID       CREATED
registry.edhat.io/rhel9/bootc-image-builder  latest    b361f3e845ea   About a minute
ago
```

4. Optional: Create a `config.toml` to configure user access, for example:

```
$ cd
$ cat config.toml
[[customizations.user]]
name = "user"
password = "pass"
key = "ssh-rsa AAA ... user@email.com"
groups = ["wheel"]
```

5. Create a directory to store the qcow2 file that we will generate for the Lab.

```
$ mkdir output
```

6. Create the QCOW2 image.

```
$ 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>
```

For other formats such as VMDK, GCE, AMI, raw and ISO, refer to Chapter 4 of Using image mode for RHEL documentation.

Chapter 4: Deploy the RHEL bootc images

You can deploy the `rhel-bootc` container image by using the following different mechanisms.

- Anaconda
- `bootc-image-builder`
- `bootc install`

The following bootc image types are available:

- Disk images that you generated by using the `bootc image-builder` such as:
  - QCOW2 (QEMU copy-on-write, virtual disk)
  - Raw (Mac Format)
  - AMI (Amazon Cloud)
  - ISO: Unattended installation method, by using an USB Sticks or Install-on-boot.

After you have created a layered image that you can deploy, there are several ways that the image can be installed to a host:

- You can use RHEL installer and Kickstart to install the layered image to a bare metal system, by using the following mechanisms:
  - Deploy by using USB
  - PXE

- You can also use `bootc-image-builder` to convert the container image to a bootc image and deploy it to a bare metal or to a cloud environment.

The installation method happens only one time. After you deploy your image, any future updates will apply directly from the container registry as the updates are published.

Lab 3: Deploying the container image by using KVM with a QCOW2 disk image

1. Create a VM using the QCOW2 file from the previous steps:

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

2. Using virt-manager, open the console to bootc vm and login using the user and password from Lab 2.4.

3. Once login use the command. Check the status of the bootc image.

```
$ sudo bootc status
```

Chapter 5: Managing RHEL bootc images

After installing and deploying RHEL bootc images, you can perform management operations on your container images, such as changing or updating the systems. The system supports in-place transactional updates with rollback after deployment.

This kind of management, also known as Day 2 management baseline, consists of transactionally fetching new operating system updates from a container registry and booting the system into them, while supporting manual, or automated rollbacks in case of failures.

You can also rely on automatic updates, that are turned on by default. The `systemd service unit` and the `systemd timer unit` files check the container registry for updates and apply them to the system. You can trigger an update process with different events, such as updating an application. There are automation tools watching these updates and then triggering the CI/CD pipelines. A reboot is required, because the updates are transactional. For environments that require more sophisticated or scheduled rollouts, you must disable auto updates and use the `bootc` utility to update your operating system.

See Day 2 operations support for more details.

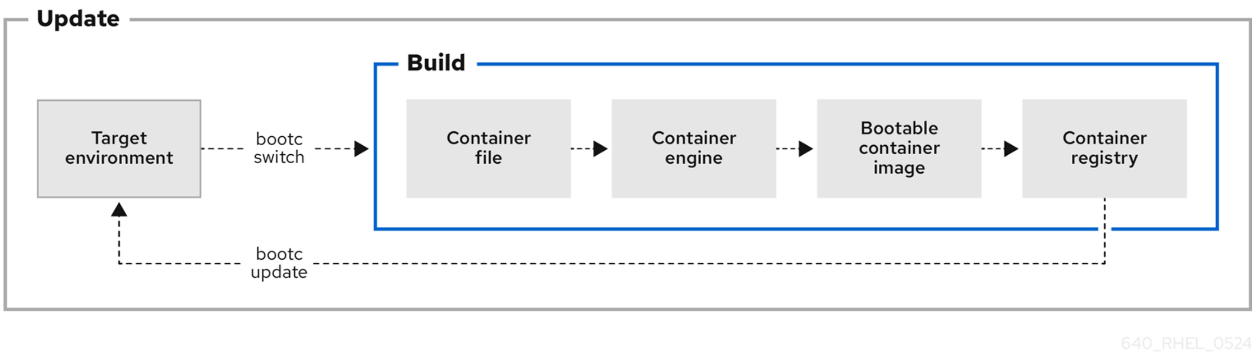

Figure 9.1. Manually updating an installed operating system, changing the container image reference or rolling back changes if needed

Lab 4: Updating bootc image

1. Modify the `Containerfile` in the webvm directory

```
$ cd webvm
$ cat Containerfile
FROM registry.redhat.io/rhel9/rhel-bootc
ADD /etc/yum.repos.d/rhel_dvd.repo /etc/yum.repos.d/
RUN dnf -y install cloud-init && \
    systemctl enable httpd && \
    dnf clean all && \
    touch /etc/mytestfile
ADD index.html /var/www/html
```

2. Build the *`webvm`* image with a new tag:

```
$ podman build -t quay.io/<namespace>/webvm:2.0 .
```

3. List all images:

```
$ podman images
REPOSITORY                    TAG      IMAGE ID       CREATED              SIZE
quay.io/<namespace>/webvm     1.0      b28cd00741b3   About 30 minutes ago 2.1 GB
quay.io/<namespace>/webvm     2.0      2cd1204a10e4   About a minute ago   2.1 GB
```

4. Push image to registry:

```
$ podman push quay.io/<namespace>/webvm:2.0
```

5. Login to quay.io in your browser using your Red Hat Login ID. Assign the web:2.0 tag as ***latest***.

Turning off automatic updates

To perform manual updates you must turn off automatic updates. You can do this by choosing one of the following options in the procedure below.

Procedure

- Disable the timer of a container build.
  - By running the `systemctl mask` command:

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

  - By modifying the `systemd` timer file. Use `systemd` "drop-ins" to override the timer. In the following example, updates are scheduled for once a week.

    1. Create an `updates.conf` file with the following content:

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

    2. Create your directory structure in systemd configuration and place your file in it:

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

Manually updating an installed operating system

To manually fetch updates from a registry and boot the system into the new updates, use `bootc upgrade`. This command fetches the transactional in-place updates from the installed operating system to the container image registry. The command queries the registry and queues an updated container image for the next boot. It stages the changes to the base image, while not changing the running system by default.

Procedure

- Run the following command:

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

  The `apply` argument is optional and you can use it when you want to automatically take actions, such as rebooting if the system has changed.

Performing rollbacks from a updated operating system

You can roll back to a previous boot entry to revert changes by using the `bootc rollback` command. This command changes the boot loader entry ordering by making the deployment under `rollback` queued for the next boot. The current deployment then becomes the rollback. Any staged changes, such as a queued upgrade that was not applied, are discarded.

After a rollback completes, the system reboots and the update timer runs within 1 to 3 hours which automatically updates and reboots your system to the image you just rolled back from.

⚠️ Warning

If you perform a rollback, the system will automatically update again unless you turn off auto-updates. See Turning off automatic updates.

Procedure

- Run the following command:

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

📔Note

The `bootc rollback` command has the same effect as `bootc upgrade`. The only difference is the container image being tracked. This enables preserving the existing states in `/etc` and `/var`, for example, host SSH keys and home directories.

Verification

- Use `systemd journal` to check the logged message for the detected rollback invocation.

```
$ journalctl -b
```

You can see a log similar to:

```
MESSAGE_ID=26f3b1eb24464d12aa5e7b544a6b5468
```

Lab 5: Manually updating an installed operating system

1. Go back to your bootc VM, and check the status now.

```
$ sudo bootc status
```

2. Manually update and reboot the machine

```
$ sudo bootc upgrade --apply
```

3. Once the system comes up, check the status and see the changes in the new image already applied..

```
$ sudo bootc status
$ ls -l /etc/mytestfile
```

4. Rollback to the previous version and check the status and changes undone.

```
$ sudo bootc rollback
$ sudo reboot
$ sudo bootc status
$ ls -l /etc/mytestfile
```