Red Hat Enterprise Linux 9.4

Using image mode for RHEL to build, deploy, and manage operating systems

Using RHEL bootable container images on Red Hat Enterprise Linux 9

Last Updated: 2024-05-14

Red Hat Enterprise Linux 9.4 Using image mode for RHEL to build, deploy, and manage operating systems

Using RHEL bootable container images on Red Hat Enterprise Linux 9

Legal Notice

Abstract

RHEL bootable container images enable you to build, deploy, and manage the operating system as if it is any other container. You can converge on a single container-native workflow to manage everything from your applications to the underlying OS.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

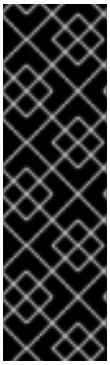
We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. INTRODUCING IMAGE MODE FOR RHEL

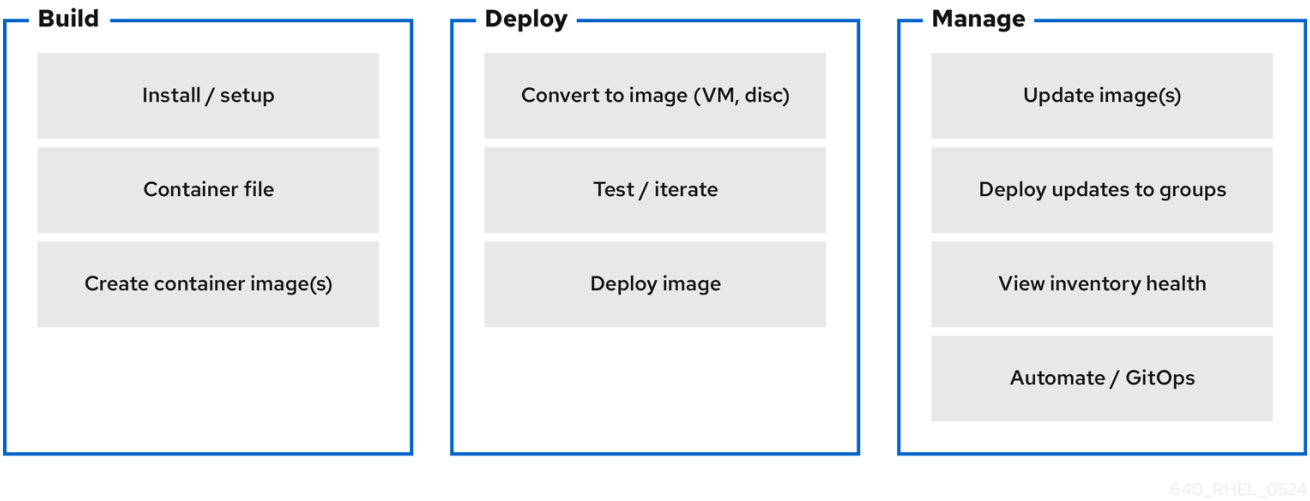
Use image mode for RHEL to build, test, and deploy operating systems by using the same tools and techniques as application containers. Image mode for RHEL is available by using the **registry.redhat.io/rhel9/rhel-bootc** bootable container image. The RHEL bootable container images differ from the existing 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-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



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 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.

Portability across hybrid cloud environments

You can use bootable container 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 bootable container images, either by using an installation mechanism, or you convert them to a disk image, the system does not run as a container.

The supported image types are the following:

- Container image formats: OCI
- Disk image formats:
  - ISO
  - QEMU copy-on-write (QCOW2), Raw
  - Amazon Machine Image (AMI)
  - Virtual Machine Image (VMI)
  - Virtual Machine Disk (VMDK)

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 bootable container 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 bootable container image contains the **bootc** utility by default and works with all image types.

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.

1. **Package mode**: the operating system uses RPM packages and is updated by using the **dnf** package manager. The root filesystem is mutable.

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 use both deployment modes to build, test, share, deploy, and manage your operating system in the same way as any other containerized application.

1.1. PREREQUISITES

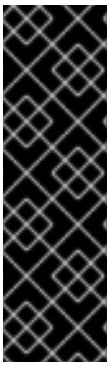
- You have a subscribed RHEL 9 system. For more information, see Getting Started with RHEL System Registration documentation.
- You have a container registry. You can create your registry locally or create a free account on the Quay.io service. To create the Quay.io account, see Red Hat Quay.io page.
- You have a Red Hat account with either production or developer subscriptions. No cost developer subscriptions are available on the Red Hat Enterprise Linux Overview page.
- You have authenticated to registry.redhat.io. For more information, see Red Hat Container Registry Authentication article.

1.2. ADDITIONAL RESOURCES

- Introducing image mode for RHEL and bootable containers in Podman Desktop quick start guide
- Image mode for Red Hat Enterprise Linux quick start: AI inference quick start guide
- Getting Started with Podman AI Lab blog article
- Customizing Anaconda product documentation
- Performing an advanced RHEL 9 installation product documentation (Kickstart)
- Composing a customized RHEL system image product documentation
- Composing, installing, and managing RHEL for Edge images product documentation

CHAPTER 2. BUILDING AND TESTING RHEL BOOTABLE CONTAINER IMAGES

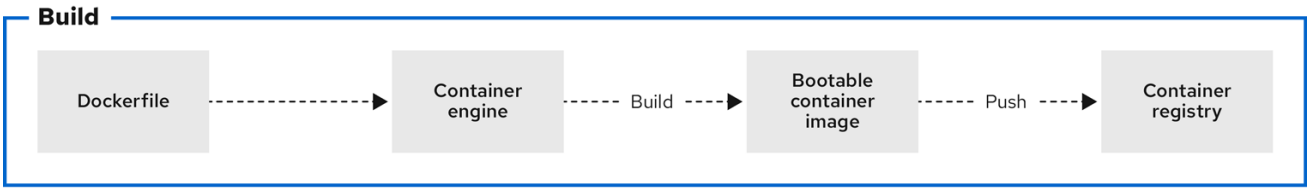
The following procedures use Podman to build and test your container image. You can also use other tools, for example, OpenShift Container Platform. For more examples of configuring RHEL systems by using containers, see the rhel-bootc-examples repository.



IMPORTANT

Red Hat provides the **rhel9/rhel-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 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 **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**.

For example, 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 bootable container to run as unprivileged users instead.

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

NOTE

Building custom **rhel-bootc** base images is not supported in this release.

2.1. BUILDING A CONTAINER IMAGE

Use the **podman build** command to build an image using instructions from a **Containerfile**.

Prerequisites

- The **container-tools** meta-package is installed.

Procedure

1. Create a **Containerfile**:

```
$ 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
```

   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.

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

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

Verification

- List all images:

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

Additional resources

- Working with container registries

- Building an image from a Containerfile with Buildah

2.2. RUNNING A CONTAINER IMAGE

Use the **podman run** command to run and test your container.

Prerequisites

- The **container-tools** meta-package is installed.

Procedure

- Run the container named **mybootc** based on the **quay.io/*<namespace>*/*<image>*:*<tag>*** container image:

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

  - The **-i** option creates an interactive session. Without the **-t** option, the shell stays open, but you cannot type anything to the shell.
  - The **-t** option opens a terminal session. Without the **-i** option, the shell opens and then exits.
  - The **--rm** option removes the **quay.io/*<namespace>*/*<image>*:*<tag>*** container image after the container exits.

Verification

- List all running containers:

```
$ 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
```

Additional resources

- Podman run command

2.3. PUSHING A CONTAINER IMAGE TO THE REGISTRY

Use the **podman push** command to push an image to your own, or a third party, registry and share it with others. The following procedure uses the Red Hat Quay registry.

Prerequisites

- The **container-tools** meta-package is installed.
- An image is built and available on the local system.
- You have created the Red Hat Quay registry. For more information see Proof of Concept - Deploying Red Hat Quay.

Procedure

Procedure

- Push the **quay.io/*<namespace>*/*<image>*:*<tag>*** container image from your local storage to the registry:

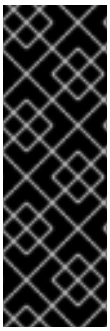
```
$ podman push quay.io/<namespace>/<image>:<tag>
```

Additional resources

- Redistributing UBI images

CHAPTER 3. CREATING BOOTC COMPATIBLE BASE DISK IMAGES WITH BOOTC-IMAGE-BUILDER

The **bootc-image-builder**, available as a Technology Preview, is a containerized tool to create disk images from bootable container images. You can use the images that you build to deploy disk images in different environments, such as the edge, server, and clouds.



IMPORTANT

Red Hat provides the **bootc-image-builder** tool 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.

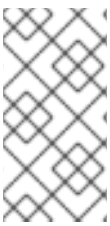
3.1. INTRODUCING IMAGE MODE FOR RHEL FOR BOOTC-IMAGE-BUILDER

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

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)

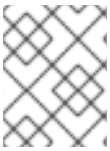
Deploying from a container image is beneficial when you run VMs or servers because you can achieve the same installation result. That consistency extends across multiple different image types and platforms when you build them from the same container image. Consequently, you can minimize the effort in 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 bootable base image. The **bootc-image-builder** tool takes the **rhel-9-bootc** OCI container image as an input.



NOTE

Building base disk images which come from private registries by using **bootc-image-builder** is not supported in this release.

Additional resources

- Red Hat products that use cloud-init

3.2. INSTALLING BOOTC-IMAGE-BUILDER

The **bootc-image-builder** is intended to be used as a container and it is not available as an RPM package in RHEL. To access it, follow the procedure.

Prerequisites

- The **container-tools** meta-package is installed. The meta-package contains all container tools, such as Podman, Buildah, and Skopeo.
- You are authenticated to **registry.redhat.io**. For details, see Red Hat Container Registry Authentication.

Procedure

1. Login to authenticate to **registry.redhat.io**:

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

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

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

Verification

- List all images pulled to your local system:

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

Additional resources

- Red Hat Container Registry Authentication
- Pulling images from registries

3.3. CREATING QCOW2 IMAGES BY USING BOOTC-IMAGE-BUILDER

Build a RHEL bootable container image into a QEMU Disk Images (QCOW2) image for the architecture that you are running the commands on.

The RHEL base image does not include a default user. Optionally, you can inject a user configuration with the **--config** option to run the bootc-image-builder container. Alternatively, you can configure the

base image with **cloud-init** to inject users and SSH keys on first boot. See Injecting users and SSH keys by using cloud-init.

Prerequisites

- You have Podman installed on your host machine.
- You have **virt-install** installed on your host machine.
- You have root access to run the **bootc-image-builder** tool, and run the containers in **--privileged** mode, to build the images.

Procedure

1. Optional: Create a **config.toml** to configure user access, for example:

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

2. Run **bootc-image-builder**. If you want to use user access configuration, pass the **config.toml** as an argument:

```
$ 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>
```

You can find the **.qcow2** image in the output folder.

Next steps

- You can deploy your image. See Deploying a container image using KVM with a QCOW2 disk image. You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

3.4. CREATING AMI IMAGES BY USING BOOTC-IMAGE-BUILDER AND UPLOADING IT TO AWS

Create an Amazon Machine Image (AMI) from a bootable container image and use it to launch an Amazon Web Service EC2 (Amazon Elastic Compute Cloud) instance.

Prerequisites

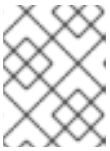
- You have Podman installed on your host machine.
- You have an existing **AWS S3** bucket within your AWS account.
- You have root access to run the **bootc-image-builder** tool, and run the containers in **--privileged** mode, to build the images.
- You have the **vmimport** service role configured on your account to import an AMI into your AWS account.

Procedure

1. Create a disk image from the bootable container image.
   - Configure the user details in the Containerfile. Make sure that you assign it with sudo access.
   - Build a customized operating system image with the configured user from the Containerfile. It creates a default user with passwordless sudo access.
2. Optional: Configure the machine image with **cloud-init**. See Injecting users and SSH keys by using cloud-init. The following is an example:

```
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}
```



NOTE

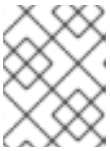
You can also use **cloud-init** to add users and additional configuration by using instance metadata.

3. Build the bootable container image. For example, to deploy the image to an **x86_64** AWS machine, use the following commands:

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

4. Use the **bootc-image-builder** tool to create an AMI from the bootc container image.

```
$ 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>
```



NOTE

The following flags must be specified all together. If you do not specify any flag, the AMI is exported to your output directory.

- **--aws-ami-name** - The name of the AMI image in AWS
- **--aws-bucket** - The target S3 bucket name for intermediate storage when you are creating the AMI
- **--aws-region** - The target region for AWS uploads
  The **bootc-image-builder** tool builds an AMI image and uploads it to your AWS s3 bucket by using your AWS credentials to push and register an AMI image after building it.

Next steps

- You can deploy your image. See Deploying a container image to AWS with an AMI disk image .
- You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

Additional resources

- AWS CLI documentation

3.5. CREATING RAW DISK IMAGES BY USING BOOTC-IMAGE-BUILDER

You can convert a bootable container image to a Raw image with an MBR or GPT partition table by using **bootc-image-builder**. The RHEL base image does not include a default user, so optionally, you can inject a user configuration with the **--config** option to run the **bootc-image-builder** container. Alternatively, you can configure the base image with **cloud-init** to inject users and SSH keys on first boot. See Injecting users and SSH keys by using cloud-init .

Prerequisites

- You have Podman installed on your host machine.
- You have root access to run the **bootc-image-builder** tool, and run the containers in **--privileged** mode, to build the images.
- You have pulled your target container image in the container storage.

Procedure

1. Optional: Create a **config.toml** to configure user access, for example:

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

2. Run **bootc-image-builder**. If you want to use user access configuration, pass the **config.toml** as an argument:

```
$ 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>
```

   You can find the **.raw** image in the output folder.

Next steps

- You can deploy your image. See Deploying a container image by using KVM with a QCOW2 disk image.
- You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

3.6. CREATING ISO IMAGES BY USING BOOTC-IMAGE-BUILDER

You can use **bootc-image-builder** to create an ISO from which you can perform an offline deployment of a bootable container.

Prerequisites

- You have Podman installed on your host machine.
- You have root access to run the **bootc-image-builder** tool, and run the containers in **--privileged** mode, to build the images.

Procedure

- Run **bootc-image-builder**:

```
$ 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>
```

You can find the **.iso** image in the output folder.

Next steps

- You can use the ISO image on unattended installation methods, such as USB sticks or Install-on-boot. The installable boot ISO contains a configured Kickstart file. See Deploying a container image by using Anaconda and Kickstart.

> **WARNING**
>
> Booting the ISO on a machine with an existing operating system or data can be destructive, because the Kickstart is configured to automatically reformat the first disk on the system.

- You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

3.7. VERIFICATION AND TROUBLESHOOTING

If you have any issues configuring the requirements for your AWS image, see the following documentation

- AWS IAM account manager
- Using high-level (s3) commands with the AWS CLI .
- S3 buckets.
- Regions and Zones.
- Launching a customized RHEL image on AWS .

For more details on users, groups, SSH keys, and secrets, see

- Managing users, groups, SSH keys, and secrets in image mode for RHEL

CHAPTER 4. DEPLOYING THE RHEL BOOTABLE IMAGES

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

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

The following bootable 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 bootable 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.

Figure 4.1. Deploying a bootable container image by using a basic build installer, bootc install, or deploying a container image by using Anaconda and Kickstart

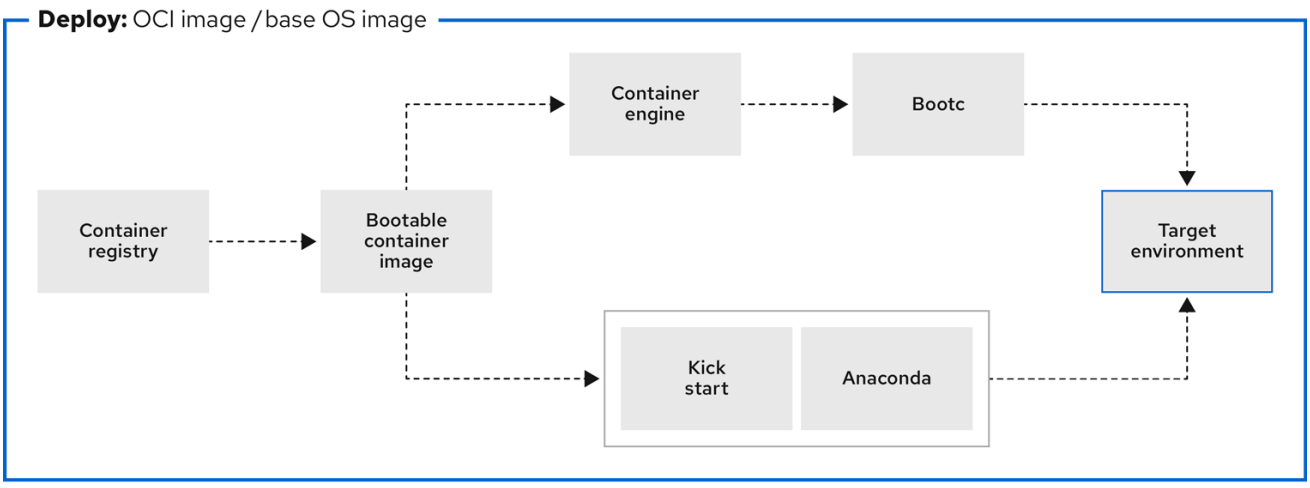
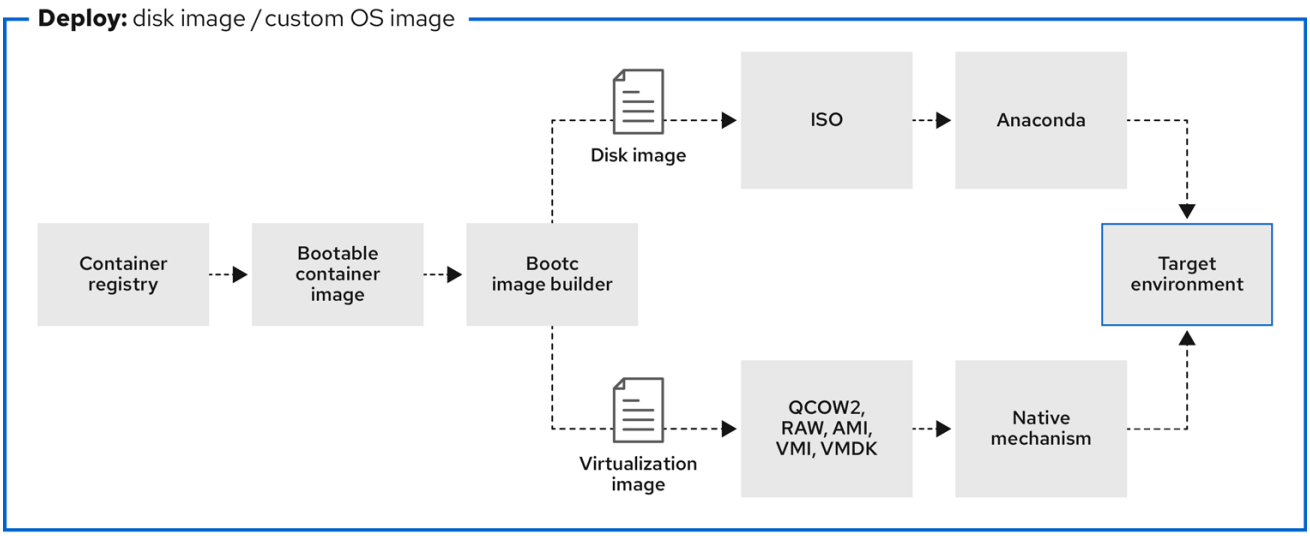


Figure 4.2. Using **bootc-image-builder** to create disk images from bootable container images and deploying disk images in different environments, such as the edge, servers, and clouds by using Anaconda, **bootc-image-builder** or **bootc install**



4.1. DEPLOYING A CONTAINER IMAGE BY USING KVM WITH A QCOW2 DISK IMAGE

After creating a QEMU disk image from a RHEL bootable container image by using the **bootc-image-builder** tool, you can use a virtualization software to boot it.

Prerequisites

- You created a container image. See Creating QCOW2 images by using bootc-image-builder .
- You pushed the container image to an accessible repository.

Procedure

- Run the container image that you create by using either **libvirt**. See Creating virtual machines by using the command-line interface for more details.
  - The following example uses **libvirt**:

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

Verification

- Connect to the VM in which you are running the container image. See Connecting to virtual machines for more details.

Next steps

- You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

Additional resources

- Configuring and managing virtualization

4.2. DEPLOYING A CONTAINER IMAGE TO AWS WITH AN AMI DISK IMAGE

After using the **bootc-image-builder** tool to create an AMI from a bootable container image, and uploading it to a AWS s3 bucket, you can deploy a container image to AWS with the AMI disk image.

Prerequisites

- You created an Amazon Machine Image (AMI) from a bootable container image. See Creating AMI images by using bootc-image-builder and uploading it to AWS.
- **cloud-init** is available in the Containerfile that you previously created so that you can create a layered image for your use case.

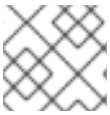
Procedure

1. In a browser, access Service→EC2 and log in.
2. On the AWS console dashboard menu, choose the correct region. The image must have the **Available** status, to indicate that it was correctly uploaded.
3. On the AWS dashboard, select your image and click **Launch**.
4. In the new window that opens, choose an instance type according to the resources you need to start your image. Click **Review and Launch**.
5. Review your instance details. You can edit each section if you need to make any changes. Click **Launch**.
6. Before you start the instance, select a public key to access it. You can either use the key pair you already have or you can create a new key pair.
7. Click **Launch Instance** to start your instance. You can check the status of the instance, which displays as **Initializing**.
   After the instance status is **Running**, the **Connect** button becomes available.
8. Click **Connect**. A window appears with instructions on how to connect by using SSH.
9. Run the following command to set the permissions of your private key file so that only you can read it. See Connect to your Linux instance .

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

10. Connect to your instance by using its Public DNS:

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



NOTE

Your instance continues to run unless you stop it.

Verification

After launching your image, you can:

- Try to connect to http://*<your_instance_ip_address>* in a browser.
- Check if you are able to perform any action while connected to your instance by using SSH.

Next steps

- After you deploy your image, you can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

Additional resources

- Pushing images to AWS Cloud AMI
- Amazon Machine Images (AMI)

4.3. DEPLOYING A CONTAINER IMAGE BY USING ANACONDA AND KICKSTART

After you convert your bootable container image to an ISO image by using **bootc-image-builder**, you can deploy the ISO image by using Anaconda and Kickstart to install your container image. The installable boot ISO already contains the **ostreecontainer** Kickstart file configured that you can use to provision your custom container image.

Prerequisites

- You have downloaded the 9.4 Boot ISO for your architecture from Red Hat. See Downloading RH boot images.

Procedure

1. Create an **ostreecontainer** Kickstart file. For example:

```
# 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. Boot a system by using the 9.4 Boot ISO installation media.
   a. Append the Kickstart file with the following to the kernel argument:

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

3. Press **CTRL+X** to boot the system.

Next steps

- After you deploy your container image, you can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

Additional resources

- ostreecontainer

4.4. DEPLOYING A CUSTOM ISO CONTAINER IMAGE

Convert a bootable container image to an ISO image by using **bootc-image-builder**. This creates a system similar to the RHEL ISOs available for download, except that your container image content is embedded in the ISO disk image. You do not need to have access to the network during installation. Then, you install the ISO disk image that you created from **bootc-image-builder** to a bare metal system.

Prerequisites

- You have created a customized container image.

Procedure

1. Create a custom installer ISO disk image with **bootc-image-builder**. See Creating ISO images by using bootc-image-builder.
2. Copy the ISO disk image to a USB flash drive.
3. Perform a bare metal installation by using the content in the USB stick into a disconnected environment.

Next steps

- After you deploy your container image, you can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

4.5. DEPLOYING AN ISO BOOTABLE CONTAINER OVER PXE BOOT

You can use a network installation to deploy the RHEL ISO image over PXE boot to run your ISO bootable container image.

Prerequisites

- You have downloaded the 9.4 Boot ISO for your architecture from Red Hat. See Downloading RH boot images.
- You have configured the server for the PXE boot. Choose one of the following options:
  - For HTTP clients, see Configuring the DHCPv4 server for HTTP and PXE boot .
  - For UEFI-based clients, see Configuring a TFTP server for UEFI-based clients .
  - For BIOS-based clients, see Configuring a TFTP server for BIOS-based clients .
- You have a client, also known as the system to which you are installing your ISO image.

Procedure

1. Export the RHEL installation ISO image to the HTTP server. The PXE boot server is now ready to serve PXE clients.
2. Boot the client and start the installation.
3. Select PXE Boot when prompted to specify a boot source. If the boot options are not displayed, press the Enter key on your keyboard or wait until the boot window opens.
4. From the Red Hat Enterprise Linux boot window, select the boot option that you want, and press Enter.
5. Start the network installation.

Next steps

- You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

Additional resources

- Preparing to install from the network using PXE
- Booting the installation from a network by using PXE

4.6. BUILDING, CONFIGURING, AND LAUNCHING DISK IMAGES WITH BOOTC-IMAGE-BUILDER

You can inject configuration into a custom image by using a Containerfile.

Procedure

1. Create a disk image. The following example shows how to add a user to the disk image.

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

- **name** - User name. Mandatory
- **password** - Nonencrypted password. Not mandatory
- **key** - Public SSH key contents. Not mandatory
- **groups** - An array of groups to add the user into. Not mandatory

2. Run **bootc-image-builder** and pass the following arguments:

```
$ 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. Launch a VM, for example, by using **virt-install**:

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

Verification

- Access the system with SSH:

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

Next steps

- After you deploy your container image, you can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

4.7. DEPLOYING A CONTAINER IMAGE BY USING BOOTC

With **bootc**, you have a container that is the source of truth. It contains a basic build installer and it is available as **bootc install to-disk** or **bootc install to-filesystem**. By using the **bootc install** methods you do not need to perform any additional steps to deploy the container image, because the container images include a basic installer.

With image mode for RHEL, you can install unconfigured images, for example, images that do not have a default password or SSH key.

Perform a bare metal installation to a device by using a RHEL ISO image.

Prerequisites

- You have downloaded the 9.4 Boot ISO for your architecture from Red Hat. See Downloading RH boot images.
- You have created a configuration file.

Procedure

- inject a configuration into the running ISO image, for example:

```
$ 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>
```

Next steps

- After you deploy your container image, you can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

4.8. ADVANCED INSTALLATION WITH TO-FILESYSTEM

The **bootc install** contains two subcommands: **bootc install to-disk** and **bootc install to-filesystem**.

- The **bootc-install-to-filesystem** performs installation to the target filesystem.
- The **bootc install to-disk** subcommand consists of a set of opinionated lower level tools that you can also call independently. The command consist of the following tools:
  - **mkfs.$fs /dev/disk**
  - **mount /dev/disk /mnt**
  - **bootc install to-filesystem --karg=root=UUID=*<uuid of /mnt>* --imgref $self /mnt**

4.8.1. Using bootc install to-existing-root

The **bootc install to-existing-root** is a variant of **install to-filesystem**. You can use it to convert an existing system into the target container image.

WARNING

This conversion eliminates the **/boot** and **/boot/efi** partitions and can delete the existing Linux installation. The conversion process reuses the filesystem, and even though the user data is preserved, the system no longer boots in package mode.

Prerequisites

- You must have root permissions to complete the procedure.

- You must match the host environment and the target container version, for example, if your host is a RHEL 9 host, then you must have a RHEL 9 container. Installing a RHEL container on a Fedora host by using **btrfs** as the RHEL kernel will not support that filesystem.

Procedure

- Run the following command to convert an existing system into the target container image. Pass the target **rootfs** by using the **-v /:/target** option.

```
# 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
```

This command deletes the data in **/boot**, but everything else in the existing operating system is not automatically deleted. This can be useful because the new image can automatically import data from the previous host system. Consequently, container images, database, the user home directory data, configuration files in **/etc** are all available after the subsequent reboot in **/sysroot**.

You can also use the **--root-ssh-authorized-keys** flag to inherit the root user SSH keys, by adding **--root-ssh-authorized-keys /target/root/.ssh/authorized_keys**. For example:

```
# 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
```

CHAPTER 5. MANAGING RHEL BOOTABLE IMAGES

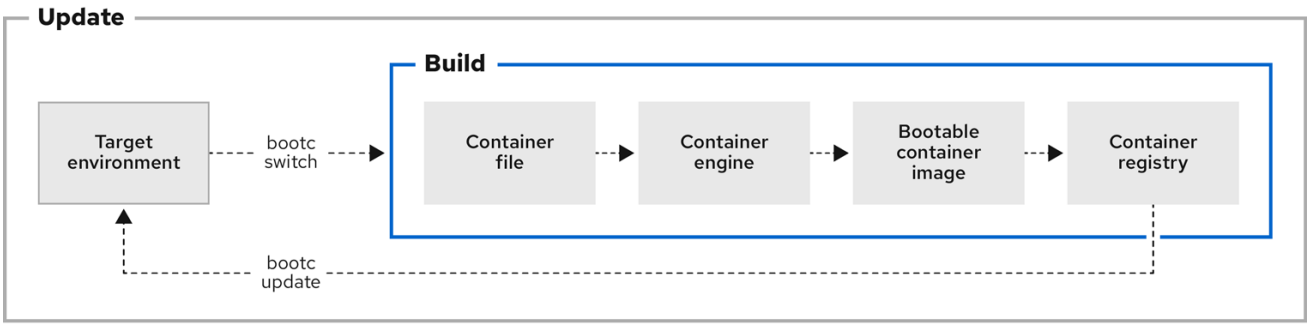
After installing and deploying RHEL bootable images, you can perform management operations on your container images, such as changing or updating the systems. The system supports in-place transactional updates with rollback after deployment.

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



5.1. SWITCHING THE CONTAINER IMAGE REFERENCE

You can change the container image reference used for upgrades by using the **bootc switch** command. For example, you can switch from the stage to the production tag. The **bootc switch** command performs the same operations as the **bootc upgrade** command and additionally changes the container image reference.

To manually switch an existing **ostree-based** container image reference, use the **bootc switch** command.

Prerequisites

- A booted system using **bootc**.

Procedure

- Run the following command:

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

Optionally, you can use the **--apply** option when you want to automatically take actions, such as rebooting if the system has changed.

NOTE

The **bootc switch** command has the same effect as **bootc upgrade**. The only difference is the container image reference is changed. This allows preserving the existing states in **/etc** and **/var**, for example, host SSH keys and home directories.

Additional resources

- The **bootc-switch** man page

5.2. PERFORMING MANUAL UPDATES FROM AN INSTALLED OPERATING SYSTEM

Installing image mode for RHEL is a one time task. You can perform any other management task, such as changing or updating the system, by pushing the changes to the container registry.

When using image mode for RHEL, you can choose to perform manual updates for your systems. Manual updates are also useful if you have an automated way to perform updates, for example, by using Ansible. Because the automatic updates are enabled by default, to perform manual updates you must turn the automatic updates off. You can do this by choosing one of the following options:

- Running the **bootc upgrade** command
- Modifying the **systemd** timer file

5.3. 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 **bootc upgrade** 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. Add your container to the file that you created:

       ```
       $ 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
       ```

5.4. 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.

NOTE

The **bootc upgrade** and **bootc update** commands are aliases.

Additional resources

- The **bootc-upgrade** man page

5.5. 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 run within 1 to 3 hours which automatically update and reboot 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 .

Prerequisites

- You performed an update to the system.

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
```

Additional resources

- The **bootc-rollback** man page

5.6. DEPLOYING UPDATES TO SYSTEM GROUPS

You can change the configuration of your operating system by modifying the Containerfile. Then you can build and push your container image to the registry. When you next boot your operating system, an update will be applied.

You can also change the container image source by using the **bootc switch** command. The container registry is the source of truth. See Switching the container image reference.

Usually, when deploying updates to system groups, you can use a central management service to provide a client to be installed on each system which connects to the central service. Often, the management service requires the client to perform a one time registration. The following is an example on how to deploy updates to system groups. You can modify it to create a persistent **systemd** service, if required.

NOTE

For clarity reasons, the Containerfile in the example is not optimized. For example, a better optimization to avoid creating multiple layers in the image is by invoking RUN a single time.

You can install a client into a image mode for RHEL image and run it at startup to register the system.

Prerequisites

- The management-client handles future connections to the server, by using a **cron** job or a separate **systemd** service.

Procedure

- Create a management service with the following characteristics. It determines when to upgrade the system.
    1. Disable **bootc-fetch-apply-updates.timer** if it is included in the base image.
    2. Install the client by using **dnf**, or some other method that applies for your client.
    3. Inject the credentials for the management service into the image.

5.7. CHECKING INVENTORY HEALTH

Health checks are one of the Day 2 Operations. You can manually check the system health of the container images and events that are running inside the container.

You can set health checks by creating the container on the command line. You can display the health check status of a container by using the **podman inspect** or **podman ps** commands.

You can monitor and print events that occur in Podman by using the **podman events** command. Each event includes a timestamp, a type, a status, a name, if applicable, and an image, if applicable.

For more information about health checks and events, see chapter Monitoring containers.

5.8. AUTOMATION AND GITOPS

You can automate the building process by using CI/CD pipelines so that an update process can be triggered by events, such as updating an application. You can use automation tools that track these updates and trigger the CI/CD pipelines. The pipeline keeps the systems up to date by using the transactional background operating system updates.

CHAPTER 6. APPENDIX: MANAGING USERS, GROUPS, SSH KEYS, AND SECRETS IN IMAGE MODE FOR RHEL

Learn more about users, groups, SSH keys, and secrets management in image mode for RHEL.

6.1. USERS AND GROUPS CONFIGURATION

RHEL image mode is a generic operating system update and configuration mechanism. You cannot use it to configure users or groups. The only exception is the **bootc install** command that has the **--root-ssh-authorized-keys** option.

Users and groups configuration for generic base images

Usually, the distribution base images do not have any configuration. Do not encrypt passwords and SSH keys with publicly-available private keys in generic images because of security risks.

Injecting SSH keys through systemd credentials

You can use **systemd** to inject a root password or SSH **authorized_keys** file in some environments. For example, use System Management BIOS (SMBIOS) to inject SSH keys system firmware. You can configure this in local virtualization environments, such as **qemu**.

Injecting users and SSH keys by using cloud-init

Many Infrastructure as a service (IaaS) and virtualization systems use metadata servers that are commonly processed by software such as **cloud-init** or **ignition**. See AWS instance metadata. The base image you are using might include **cloud-init** or Ignition, or you can install it in your own derived images. In this model, the SSH configuration is managed outside of the bootable image.

Adding users and credentials by using container or unit custom logic

Systems such as **cloud-init** are not privileged. You can inject any logic you want to manage credentials in the way you want to launch a container image, for example, by using a **systemd** unit. To manage the credentials, you can use a custom network-hosted source, for example, FreeIPA.

Adding users and credentials statically in the container build

In package-oriented systems, you can use the derived build to inject users and credentials by using the following command:

```
RUN useradd someuser
```

You can find issues in the default **shadow-utils** implementation of **useradd**: Users and groups IDs are allocated dynamically, and this can cause drift.

User and group home directories and /var directory

For systems configured with persistent **/home → /var/home**, any changes to **/var** made in the container image after initial installation will not be applied on subsequent updates.
For example, if you inject **/var/home/someuser/.ssh/authorized_keys** into a container build, existing systems do not get the updated **authorized_keys** file.

Using DynamicUser=yes for systemd units

Use the **systemd DynamicUser=yes** option where possible for system users.
This is significantly better than the pattern of allocating users or groups at package install time, because it avoids potential UID or GID drift.

Using systemd-sysusers

Use **systemd**-sysusers, for example, in your derived build. For more information, see the **systemd**-sysusers documentation.

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

The **sysusers** tool makes changes to the traditional **/etc/passwd** file as necessary during boot time. If **/etc** is persistent, this can avoid **UID** or **GID** drift. It means that the **UID** or **GID** allocation depends on how a specific machine was upgraded over time.

Using systemd JSON user records

See JSON user records **systemd** documentation. Unlike **sysusers**, the canonical state for these users lives in **/usr**. If a subsequent image drops a user record, then it also vanishes from the system.

Using nss-altfiles

With **nss-altfiles**, you can remove the **systemd** JSON user records. It splits system users into **/usr/lib/passwd** and **/usr/lib/group**, aligning with the way the OSTree project handles the 3 way merge for **/etc** as it relates to **/etc/passwd**. Currently, if the **/etc/passwd** file is modified in any way on the local system, then subsequent changes to **/etc/passwd** in the container image are not applied. Base images built by **rpm-ostree** have **nns-altfiles** enabled by default.

Also, base images have a system users pre-allocated and managed by the NSS file to avoid UID or GID drift.

In a derived container build, you can also append users to **/usr/lib/passwd**, for example. Use **sysusers.d** or **DynamicUser=yes**.

Machine-local state for users

The filesystem layout depends on the base image.
By default, the user data is stored in both **/etc**, **/etc/passwd**, **/etc/shadow** and **groups**, and **/home**, depending on the base image. However, the generic base images have to both be machine-local persistent state. In this model **/home** is a symlink to **/var/home/*user***.

Injecting users and SSH keys at system provisioning time

For base images where **/etc** and **/var** are configured to persist by default, you can inject users by using installers such as Anaconda or Kickstart.
Typically, generic installers are designed for one time bootstrap. Then, the configuration becomes a mutable machine-local state that you can change in Day 2 operations, by using some other mechanism.

You can use the Anaconda installer to set the initial password. However, changing this initial password requires a different in-system tool, such as **passwd**.

These flows work equivalently in a **bootc-compatible** system, to support users directly installing generic base images, without requiring changes to the different in-system tool.

Transient home directories

Many operating system deployments minimize persistent, mutable, and executable state. This can damage user home directories.
The **/home** directory can be set as **tmpfs**, to ensure that user data is cleared across reboots. This approach works especially well when combined with a transient **/etc** directory.

To set up the user's home directory to, for example, inject SSH **authorized_keys** or other files, use the **systemd tmpfiles.d** snippets:

-

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

SSH is embedded in the image as: **/usr/lib/tmpfiles.d/<username-keys.conf**. Another example is a service embedded in the image that can fetch keys from the network and write them. This is the pattern used by **cloud-init**.

UID and GID drift

The **/etc/passwd** and similar files are a mapping between names and numeric identifiers. When the mapping is dynamic and mixed with "stateless" container image builds, it can cause issues. Each container image build might result in the UID changing due to RPM installation ordering or other reasons. This can be a problem if that user maintains a persistent state. To handle such cases, convert it to use **sysusers.d** or use **DynamicUser=yes**.

6.2. INJECTING SECRETS IN IMAGE MODE FOR RHEL

Image mode for RHEL does not have an opinionated mechanism for secrets. You can inject container pull secrets in your system for some cases, for example:

- For **bootc** to fetch updates from a registry that requires authentication, you must include a pull secret in a file. In the following example, the **creds** secret contains the registry pull secret.

```
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
```

  To build it, run **podman build --secret id=creds,src=$HOME/.docker/config.json**. Use a single pull secret for **bootc** and Podman by using a symlink to both locations to a common persistent file embedded in the container image, for example **/usr/lib/container-auth.json**.

- For Podman to fetch container images, include a pull secret to **/etc/containers/auth.json**. With this configuration, the two stacks share the **/usr/lib/container-auth.json** file.

  #### Injecting secrets by embedding them in a container build

  You can include secrets in the container image if the registry server is suitably protected. In some cases, embedding only bootstrap secrets into the container image is a viable pattern, especially alongside a mechanism for having a machine authenticate to a cluster. In this pattern, a provisioning tool, whether run as part of the host system or a container image, uses the bootstrap secret to inject or update other secrets, such as SSH keys, certificates, among others.

  #### Injecting secrets by using cloud metadata

  Most production Infrastructure as a Service (IaaS) systems support a metadata server or equivalent which can securely host secrets, particularly bootstrap secrets. Your container image can include tools such as **cloud-init** or **ignition** to fetch these secrets.

  #### Injecting secrets by embedding them in disk images

  You can embed **bootstrap secrets** only in disk images. For example, when you generate a cloud disk image from an input container image, such as AMI or OpenStack, the disk image can contain secrets that are effectively machine-local state. Rotating them requires an additional management tool or refreshing the disk images.

  #### Injecting secrets by using bare metal installers

Installer tools usually support injecting configuration through secrets.

Injecting secrets through systemd credentials

The **systemd** project has a credential concept for securely acquiring and passing credential data to systems and services, which applies in some deployment methodologies. See the systemd credentials documentation for more details.

Additional resources

- See Example bootable containers.