Red Hat Enterprise Linux 10

Composing, installing, and managing RHEL for Edge images

Creating, deploying, and managing Edge systems with RHEL 10

Last Updated: 2025-06-10

Red Hat Enterprise Linux 10 Composing, installing, and managing RHEL for Edge images

Creating, deploying, and managing Edge systems with RHEL 10

Legal Notice

Abstract

You can use image mode for RHEL to build operating system images suitable for your edge deployments. Then, you can remotely install, and securely manage and scale deployments of the images on Edge servers.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 3

CHAPTER 1. INTRODUCING RHEL FOR EDGE IMAGES 4
1.1. DIFFERENCE BETWEEN RHEL RPM IMAGES AND RHEL FOR EDGE IMAGES 4
1.2. ADDITIONAL RESOURCES 5

CHAPTER 2. MIGRATING FROM RPM-OSTREE-BASED DEPLOYED SYSTEMS TO BOOTC-BASED SYSTEMS 6
2.1. IMAGE MODE FOR RHEL 6
2.2. BUILDING CUSTOMIZED IMAGES FROM A BASE IMAGE 6
2.3. EQUIVALENCE BETWEEN BLUEPRINT CUSTOMIZATIONS AND CONTAINERFILE CUSTOMIZATIONS 7
2.3.1. Creating similar RHEL for Edge images by using image mode for RHEL 9
2.4. BUILDING CUSTOMIZED IMAGES BY USING PODMAN BUILD 11
2.4.1. Using image mode to create a 9.6 RHEL for Edge image 11
2.4.2. Using image mode to create a RHEL 10 for Edge image 13
2.5. USING BOOTC-IMAGE-BUILDER TO BUILD DISK IMAGES BASED ON A CONTAINER 15
2.5.1. Installing bootc-image-builder 15
2.5.2. Using bootc-image-builder to RHEL 9.6 disk images 16
2.5.3. Using bootc-image-builder to create RHEL 10.0 disk images 17
2.6. SWITCHING FROM AN EXISTING RPM-OSTREE INSTALLATION TO IMAGE MODE FOR RHEL 9.6 19
2.6.1. Adding existing system groups and users information to a Containerfile 19
2.6.2. Switching a 9.6 RHEL for Edge installed by using a raw image to image mode 21
2.6.3. Switching a 9.6 RHEL for Edge installed by using a simplified-installer image to image mode 22
2.7. UPGRADING FROM AN EXISTING RPM-OSTREE INSTALLATION TO IMAGE MODE FOR RHEL 10.0 23
2.7.1. Upgrading an existing RHEL for Edge system to image mode for RHEL 10.0 23

CHAPTER 3. AUTOMATICALLY PROVISIONING AND ONBOARDING RHEL FOR EDGE DEVICES WITH FDO . 25
3.1. THE FIDO DEVICE ONBOARDING (FDO) PROCESS 25
3.2. FDO AUTOMATIC ONBOARDING TECHNOLOGIES 27
3.3. AUTOMATICALLY PROVISIONING AND ONBOARDING RHEL FOR EDGE DEVICES 28
3.4. GENERATING KEY AND CERTIFICATES 29
3.5. INSTALLING AND RUNNING THE MANUFACTURING SERVER 30
3.6. INSTALLING, CONFIGURING, AND RUNNING THE RENDEZVOUS SERVER 32
3.7. INSTALLING, CONFIGURING, AND RUNNING THE OWNER SERVER 33
3.8. AUTOMATICALLY ONBOARDING A RHEL FOR EDGE DEVICE BY USING FDO AUTHENTICATION 35
3.9. DEPLOYING AN IMAGE MODE FOR RHEL SYSTEMS BY USING FDO 36

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 RHEL FOR EDGE IMAGES

A RHEL for Edge image is an **rpm-ostree** image that includes system packages to remotely install RHEL on Edge servers.

The system packages include:

- **Base OS** package
- Podman as the container engine
- Additional RPM Package Manager (RPM) content

Differently from RHEL images, RHEL for Edge is an immutable operating system, that is, it contains a **read-only** root directory with the following characteristics:

- The packages are isolated from the root directory.
- Each version of the operating system is a separate deployment. Therefore, you can roll back the system to a previous deployment when needed.
- Offers efficient updates over the network.
- Supports multiple operating system branches and repositories.
- Contains a hybrid **rpm-ostree** package system

You can deploy a RHEL for Edge image on Bare Metal, Appliance, and Edge servers.

With a RHEL for Edge image, you can achieve the following benefits:

Atomic upgrades

You know the state of each update, and no changes are seen until you reboot your system.

Custom health checks and intelligent rollbacks

You can create custom health checks, and if a health check fails, the operating system rolls back to the previous stable state.

Container-focused workflow

The image updates are staged in the background, minimizing any workload interruptions to the system.

Optimized Over-the-Air updates

You can make sure that your systems are up-to-date, even with intermittent connectivity, thanks to efficient over-the-air (OTA) delta updates.

1.1. DIFFERENCE BETWEEN RHEL RPM IMAGES AND RHEL FOR EDGE IMAGES

You can create RHEL system images in traditional package-based RPM format and also as RHEL for Edge (**rpm-ostree**) images.

You can use the traditional package-based RPMs to deploy RHEL on traditional data centers. However, with RHEL for Edge images you can deploy RHEL on servers other than traditional data centers. These servers include systems where processing of large amounts of data is done closest to the source where data is generated, the Edge servers.

The RHEL for Edge (**rpm-ostree**) images are not a package manager. They only support complete bootable file system trees, not individual files. These images do not have information regarding the individual files such as how these files were generated or anything related to their origin.

The **rpm-ostree** images need a separate mechanism, the package manager, to install additional applications in the **/var** directory. With that, the **rpm-ostree** image keeps the operating system unchanged, while maintaining the state of the **/var** and **/etc** directories. The atomic updates enable rollbacks and background staging of updates.

Refer to the following table to know how RHEL for Edge images differ from the package-based RHEL RPM images.

Table 1.1. Difference between RHEL RPM images and RHEL for Edge images

Key attributes	RHEL RPM image	RHEL for Edge image
OS assembly	You can assemble the packages locally to form an image.	The packages are assembled in an OSTree which you can install on a system.
OS updates	You can use **dnf update** to apply the available updates from the enabled repositories.	You can use **rpm-ostree upgrade** to stage an update if any new commit is available in the OSTree remote at **/etc/ostree/remotes.d/**. The update takes effect on system reboot.
Repository	The package contains DNF repositories	The package contains OSTree remote repository
User access permissions	Read write	Read-only (**/usr**)
Data persistence	You can mount the image to any non **tmpfs** mount point	**/etc** & **/var** are read/write enabled and include persisting data.

1.2. ADDITIONAL RESOURCES

- Interactively installing RHEL from installation media

CHAPTER 2. MIGRATING FROM RPM-OSTREE-BASED DEPLOYED SYSTEMS TO BOOTC-BASED SYSTEMS

Starting with RHEL 10.0, you can no longer use RHEL image builder to build RHEL for Edge images. RHEL 10.0 and later versions no longer support using OSbuild to build edge artifacts. Instead, you can use image mode for RHEL to build operating system images suitable for your edge deployments. If you prefer, you can continue to use RHEL image builder on RHEL 9 to build RHEL for Edge artifacts.

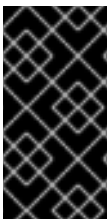
To use image mode for RHEL, you can upgrade from RHEL 9 image builder to image mode for RHEL 10 and use image mode for RHEL to build bootable container images that you can use for your Edge deployments.

The image mode for RHEL feature enables you to customize your operating system with the container image located at **registry.redhat.io/rhel9/rhel-bootc**. You can also build a smaller bootc base image from scratch similar in size and content to the basic RHEL for the Edge OSTree commit.

2.1. IMAGE MODE FOR RHEL

Image mode for Red Hat Enterprise Linux (RHEL) is a deployment method that uses a container-native approach to build, deploy, and manage the operating system as a bootc base image (rhel-bootc). The bootc base image (rhel-bootc) contains the necessary components for a bootable operating system such as kernel, firmware, boot loader, among others. You can use it to build, deploy, and manage the operating system as if it is any other container.

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/rhel10/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

Image mode for RHEL does not support **rpm-ostree** file system with blueprint customization. You cannot build disk images from bootc images by using **osbuild-composer**. Instead, use **bootc-image-builder** to generate disk images from bootc images.

Additional resources

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

2.2. BUILDING CUSTOMIZED IMAGES FROM A BASE IMAGE

You can use Podman to build and test your container image. A general **Containerfile** has the following structure:

```
FROM registry.redhat.io/rhel10/rhel-bootc:latest

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

ADD [application]
```

> ADD [configuration files]
>
> RUN [config scripts]

For more examples of configuring RHEL systems by using containers, see the rhel-bootc-examples repository.

2.3. EQUIVALENCE BETWEEN BLUEPRINT CUSTOMIZATIONS AND CONTAINERFILE CUSTOMIZATIONS

The following table contains the blueprint customization options and the equivalent command to be used in the Containerfile. .Table

Blueprint	Command instructions
distro = "rhel-10."	FROM rhel-bootc:10
[[packages]] name = "openssh-server" version = "8.*"	RUN dnf install *<package name>*
[[groups]] name = "anaconda-tools"	RUN dnf group install *<group_name>*
[[containers]] source = "quay.io/rhel/rhel:latest"	RUN podman pull docker.io/library/postgres:alpine
[customizations.kernel] name = "kernel-debug" append = "nosmt=force"	RUN mkdir -p /usr/lib/bootc/kargs.d RUN cat <<`EOF` >> /usr/lib/bootc/kargs.d/console.toml kargs = ["console=ttyS0,114800n8","kernel-debug"] match-architectures = ["x86_64"] **EOF**
[customizations.rhsm.config.dnf_plugins.product_id] enabled = true [customizations.rhsm.config.dnf_plugins.subscription_manager] enabled = true [customizations.rhsm.config.subscription_manager.rhsm] manage_repos = true [customizations.rhsm.config.subscription_manager.rhsmcertd] auto_registration = true	COPY ./rhsm.conf /etc/rhsm/rhsm.conf

Blueprint	Command instructions
[customizations.rpm.import_keys] files = ["/etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-18-primary", "/etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-19-primary"]	RUN mkdir -p /etc/pki/rpm-gpg/ COPY <host_path>/gpg_key /etc/pki/rpm-gpg//gpg_key
[[customizations.sshkey]] user = "root" key = "PUBLIC SSH KEY"	# SSH keys COPY test.pub container_key.pub RUN mkdir -p .ssh && \ cat container_key.pub >> .ssh/authorized_keys && \ chmod 600 .ssh/authorized_keys && \ rm -f container_path_to_key.pub
[customizations.timezone] timezone = "US/Eastern" ntpservers = ["0.north-america.pool.ntp.org", "1.north-america.pool.ntp.org"]	RUN ln -sf /usr/share/zoneinfo/Asia/Bangkok /etc/localtime
[customizations.locale] languages = ["en_US.UTF-8"] keyboard = "us"	RUN cat <<`EOF` >> /etc/locale.conf LANG="en_US.UTF-8" **EOF** && \ cat <<`EOF` >> /etc/vconsole.conf KEYMAP=us **EOF**
[customizations.firewall] ports = ["22:tcp", "80:tcp", "imap:tcp", "53:tcp", "53:udp", "30000-32767:tcp", "30000-32767:udp"]	RUN dnf install -y firewalld && \ dnf clean all && \ firewall-offline-cmd --new-zone=customzone && \ firewall-offline-cmd --zone=customzone --set-description="Custom firewall rules for the container" && \ firewall-offline-cmd --zone=customzone --add-service=ftp && \ firewall-offline-cmd --zone=customzone --add-service=ntp && \ firewall-offline-cmd --zone=customzone --add-service=dhcp && \ firewall-offline-cmd --zone=customzone --add-port=22/tcp && \ firewall-offline-cmd --zone=customzone --add-port=80/tcp && \ firewall-offline-cmd --zone=customzone --add-port=53/tcp && \ firewall-offline-cmd --zone=customzone --add-port=53/udp && \ firewall-offline-cmd --zone=customzone --add-port=30000-32767/tcp && \ firewall-offline-cmd --zone=customzone --add-port=30000-32767/udp && \ firewall-offline-cmd --set-default-zone=customzone
[[customizations.directories]] path = "/etc/*<dir-name>*" mode = "0755" user = "root" group = "root" ensure_parents = false	#Directory: RUN mkdir /etc/*<dir>* RUN chown -R admin:wheel /etc/*<dir>* && \ chmod -R 644 /etc/*<dir>* #Files: RUN touch /etc/*<myfile>* RUN chown :widget /etc/*<myfile>* && \ chmod 600 /etc/*<myfile>*
[customizations] installation_device = "/dev/sda"	RUN mkdir -p /usr/lib/bootc/kargs.d && \ cat <<`EOF` >> /usr/lib/bootc/kargs.d/console.toml kargs = ["inst.device=/dev/sda"] **EOF**

Blueprint	Command instructions
[customizations.ignition.embedded] config = "eyJpZ25pdG....xln1dfX0="	RUN mkdir -p /usr/lib/bootc/kargs.d && \ cat <<`EOF` >> /usr/lib/bootc/kargs.d/console.toml kargs = ["ignition.config.url=http://192.168.122.1/fiot.ign","rd.neednet=1"] **EOF**
[customizations.fdo] manufacturing_server_url = "http://192.168.122.199:8080" diun_pub_key_insecure = "true" di_mfg_string_type_mac_iface = "enp2s0"	RUN dnf install -y fdo-init fdo-client && \ systemctl enable fdo-client-linuxapp.service
[customizations.openscap] datastream = "/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml" profile_id = "xccdf_org.ssgproject.content_profile_cis" [customizations.openscap.json_tailoring] profile_id = "*<name-of-profile-used-in-json-tailoring>*-file" filepath = "*/some/path/tailoring-file*.json" [[customizations.files]] path = "*/the/path/tailoring-file*.json" data = "*<json-tailoring-file-contents>*"	RUN dnf install -y openscap-utils && \ autotailor --output /some/path/tailoring-file.json \ --new-profille-id xccdf_org.ssgproject.content_profile_cis
[customizations] fips = true	RUN mkdir -p /usr/lib/bootc/kargs.d && \ cat <<`EOF` >> /usr/lib/bootc/kargs.d/01-fips.toml kargs = ["fips=1"] **EOF** && \ update-crypto-policies --no-reload --set FIPS

2.3.1. Creating similar RHEL for Edge images by using image mode for RHEL

To create edge images by using image mode for RHEL, you must manually install some missing packages that are common to an OSTree commit. Most of these packages are part of the bootc images, but there are a few missing packages, such as:

- **clevis**
- **clevis-dracut**
- **clevis-luks**
- **greenboot**
- **greenboot-default-health-checks**

- **fdo-client**
- **fdo-owner-cli**

To install the missing packages and create your similar RHEL for Edge image, follow these steps:

Prerequisites

- An existing RHEL for Edge rpm-ostree based deployed system.

Procedure

1. Create a Containerfile with the following content:

```
FROM registry.io.redhat.com/rhel10/rhel-bootc:latest

RUN dnf install -y  \
    clevis  \
    clevis-dracut  \
    clevis-luks  \
    greenboot  \
    greenboot-default-health-checks  \
    fdo-client  \
    fdo-owner-cli
# (Optionl) Extra packages often used in edge
# RUN dnf install -y
#     dracut-config-generic  \
#     platform-python  \
#     pinentry  \
#     firewalld  \
#     iptables  \
#     NetworkManager-wifi  \
#     NetworkManager-wwan  \
#     wpa_supplicant  \
#     traceroute  \
#     rootfiles  \
#     policycoreutils-python-utils  \
#     setools-console  \
#     rsync  \
#     usbguard
RUN systemctl enable NetworkManager.service  \
    greenboot-grub2-set-counter.service  \
    greenboot-grub2-set-success.service  \
    greenboot-healthcheck.service  \
    greenboot-rpm-ostree-grub2-check-fallback.service  \
    greenboot-status.service  \
    greenboot-task-runner.service
    redboot-auto-reboot.service  \
    redboot-task-runner.service"
```

2. Build your similar RHEL for Edge customized bootc image:

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

3. Optional: push the image:

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

Verification

- List all images:

```
$ podman images
```

2.4. BUILDING CUSTOMIZED IMAGES BY USING PODMAN BUILD

Starting with RHEL 9.6, you can continue to use RHEL image builder to create edge installation images. You can also use image mode for RHEL to compose container images and create disk images for deployment. If you want to continue to use RHEL image builder, see Composing, installing, and managing RHEL for Edge images.

To use image mode for RHEL to create new disk images, follow the steps:

2.4.1. Using image mode to create a 9.6 RHEL for Edge image

To build an image mode RHEL for an edge host, create a Containerfile with instructions. Then use **bootc-image-builder** to install the created edge host by using an anaconda ISO.

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

1. Create a **Containerfile**, for example:

```
$ cat Containerfile
FROM registry.redhat.io/rhel9/rhel-bootc:9.6

# Packages
RUN dnf install -y zsh && dnf clean all

# Group install
RUN dnf group -y install "Development Tools"

# Hostname
RUN echo "rock.paper.scissor" > /etc/hostname

# Kernel
RUN mkdir -p /usr/lib/bootc/kargs.d
RUN cat <<EOF >> /usr/lib/bootc/kargs.d/console.toml
kargs = ["console=ttyS0,114800n8","kernel-debug"]
match-architectures = ["x86_64"]
EOF

# Subscription-manager
```

```
RUN dnf install subscription-manager

# RPM config
RUN mkdir -p /etc/pki/rpm-gpg/
COPY <host_path>/gpg_key /etc/pki/rpm-gpg/gpg_key

# Timezones
RUN cat <<EOF >> /etc/localtime
Asia/Bangkok
EOF


# Locale
RUN cat <<EOF >> /etc/locale.conf
LANG="en_US.UTF-8"
EOF && \
cat <<EOF >> /etc/vconsole.conf
KEYMAP=us
EOF


# firewall
RUN dnf install -y firewalld && \
    mkdir -p /etc/firewalld/zones
RUN cat <<EOF >> /etc/firewalld/zones/customzone.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
    <short>Customzone</short>
    <description>Custom firewall rules for the container.</description>

    <!-- Allowed services -->
    <service name="ftp"/>
    <service name="ntp"/>
    <service name="dhcp"/>

    <!-- Blocked services (not explicitly listed) -->
    <!-- Removing telnet explicitly is unnecessary if it is not included -->

    <!-- Open specific ports -->
    <port protocol="tcp" port="22"/>
    <port protocol="tcp" port="80"/>
    <port protocol="tcp" port="53"/>
    <port protocol="udp" port="53"/>
    <port protocol="tcp" port="30000-32767"/>
    <port protocol="udp" port="30000-32767"/>
</zone>
EOF
RUN firewall-offline-cmd --set-default-zone=customzone

# systemd services
RUN systemctl enable sshd

#ignition
RUN mkdir -p /usr/lib/bootc/kargs.d && \
cat <<EOF >> /usr/lib/bootc/kargs.d/console.toml
kargs = ["ignition.config.url=http://192.168.122.1/fiot.ign","rd.neednet=1"]
```

```
EOF

#fdo
RUN dnf install -y fdo-init fdo-client && \
systemctl enable fdo-client-linuxapp.service

#Repositories
RUN mkdir -p /etc/yum.repos.d
COPY custom.repo /etc/yum.repos.d/custom.repo

#fips
RUN mkdir -p /usr/lib/bootc/kargs.d && \
cat <<EOF >>  /usr/lib/bootc/kargs.d/01-fips.toml
kargs = ["fips=1"]
EOF
RUN dnf install -y crypto-policies-scripts && update-crypto-policies --no-reload --set FIPS
```

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

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

Verification

- List all images:

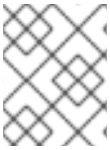
```
$ podman images
REPOSITORY                    TAG      IMAGE ID      CREATED             SIZE
quay.io/<namespace>/<image>   latest   b28cd00741b3  About a minute ago  2.1 GB
```

Additional resources

- Working with container registries
- Managing users, groups, SSH keys, and secrets in image mode for RHEL documentation

2.4.2. Using image mode to create a RHEL 10 for Edge image

From RHEL 10 and later, to create new RHEL for Edge images installations, use **bootc**, because RHEL image builder no longer supports edge artifacts.



NOTE

Not all the available RHEL image builder artifacts are available in image mode. That means that you cannot create certain image types by using **bootc-image-builder**.

- Notably, the **simplified-installer** no longer exists. Instead, use the **bootc-image-builder** Anaconda ISO for workflows such as FDO.

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

1. Create a **Containerfile**. The following example contains several customizations that you can use as an example, and can be removed in case it does not suit your requirements.

```
$ cat Containerfile
FROM registry.redhat.io/rhel10/rhel-bootc:10.0

# Packages
RUN dnf install -y zsh && dnf clean all

# Group install
RUN dnf group -y install "Development Tools"

# Kernel
RUN mkdir -p /usr/lib/bootc/kargs.d
RUN cat <<EOF >> /usr/lib/bootc/kargs.d/console.toml
kargs = ["console=ttyS0,114800n8","kernel-debug"]
match-architectures = ["x86_64"]
EOF

# Subscription-manager
COPY ./rhsm.conf /etc/rhsm/rhsm.conf

# RPM config
RUN mkdir -p /etc/pki/rpm-gpg/
COPY <host_path>/gpg_key /etc/pki/rpm-gpg//gpg_key

# Additional groups
RUN groupadd -g 1001 widget

# Timezones
RUN ln -sf /usr/share/zoneinfo/Asia/Bangkok /etc/localtime

# Locale
RUN cat <<EOF >> /etc/locale.conf
LANG="en_US.UTF-8"
EOF && \
cat <<EOF >> /etc/vconsole.conf
KEYMAP=us
EOF


# firewall
RUN dnf install -y firewalld && \
    dnf clean all && \
    firewall-offline-cmd --new-zone=customzone && \
    firewall-offline-cmd --zone=customzone --set-description="Custom firewall rules for the
container" && \
    firewall-offline-cmd --zone=customzone --add-service=ftp && \
    firewall-offline-cmd --zone=customzone --add-service=ntp && \
    firewall-offline-cmd --zone=customzone --add-service=dhcp && \
    firewall-offline-cmd --zone=customzone --add-port=22/tcp && \
```

```
    firewall-offline-cmd --zone=customzone --add-port=80/tcp && \
    firewall-offline-cmd --zone=customzone --add-port=53/tcp && \
    firewall-offline-cmd --zone=customzone --add-port=53/udp && \
    firewall-offline-cmd --zone=customzone --add-port=30000-32767/tcp && \
    firewall-offline-cmd --zone=customzone --add-port=30000-32767/udp && \
    firewall-offline-cmd --set-default-zone=customzone

# systemd services
RUN systemctl enable httpd sshd && \
systemctl disable telnetd && \
systemctl mask rcpbind
```

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
quay.io/<namespace>/<image>   latest   b28cd00741b3  About a minute ago  2.1 GB
```

Additional resources

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

2.5. USING BOOTC-IMAGE-BUILDER TO BUILD DISK IMAGES BASED ON A CONTAINER

You can use **bootc-image-builder** to create disk images based on a container.

2.5.1. 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/rhel10/bootc-image-builder
```

Verification

- List all images pulled to your local system:

```
$ sudo podman images
REPOSITORY                                  TAG         IMAGE ID      CREATED       SIZE
registry.redhat.io/rhel10/bootc-image-builder  latest      b361f3e845ea  24 hours ago  676 MB
```

Additional resources

- Red Hat Container Registry Authentication
- Pulling images from registries

2.5.2. Using bootc-image-builder to RHEL 9.6 disk images

You can use **bootc-image-builder** to create a disk images that can be used to deploy a bootable container to your host.

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

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

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

2. Manually pull the image:

```
$ sudo podman pull quay.io/quay.io/<_namespace_>/<_image_>:<_tag_>
```

3. Create the **output** directory for the image that you are building:

```
$ mkdir output
```

4. Run **bootc-image-builder** to create the image. If you do not want to add any configuration, omit the **-v $(pwd)/config.toml:/config.toml** 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 $(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.

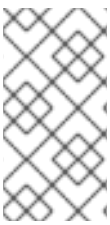
WARNING

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.

2.5.3. Using bootc-image-builder to create RHEL 10.0 disk images

Starting with RHEL 10 and later, RHEL image builder no longer supports composing customized RHEL **rpm-ostree** images optimized for Edge. To create new RHEL images for Edge environments as part of RHEL 10, you must use image mode for RHEL.



NOTE

Not all the available RHEL image builder artifacts are available in image mode. That means that you cannot create certain image types by using **bootc-image-builder**. The **simplified-installer** no longer exists. Instead, use the **bootc-image-builder** Anaconda ISO for FDO workflow.

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

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

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

2. Manually pull the image:

```
$ sudo podman pull quay.io/<namespace>/<image>:_<tag>_
```

3. Create the **output** directory for the image that you are building:

```
$ mkdir output
```

4. Run **bootc-image-builder** to create the image. If you do not want to add any configuration, omit the **-v $(pwd)/config.toml:/config.toml** 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 $(pwd)/config.toml:/config.toml \
  -v $(pwd)/output:/output \
  registry.redhat.io/rhel10/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 image on a machine with an existing operating system or data can be destructive and cause data loss.

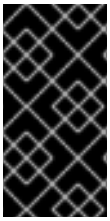
- You can make updates to the image and push the changes to a registry. See Managing RHEL bootable images.

2.6. SWITCHING FROM AN EXISTING RPM-OSTREE INSTALLATION TO IMAGE MODE FOR RHEL 9.6

You can use image mode for RHEL on your existing RHEL for Edge system by using the **bootc switch** command.

2.6.1. Adding existing system groups and users information to a Containerfile

When switching between different host builds, you can use the **bootc switch** command to migrate your existing deployed system to a bootc based system.



IMPORTANT

Some user and group IDs differ between **rhel-bootc images** and RHEL for Edge. This affects several groups and users, such as **ssh_keys**. As a consequence, the private keys belong to a group with a misconfigured ID, and you will not be able to use public keys to access the edge system.

The Image Mode system uses **altfiles** to manage users at **/user/lib/passwd** and groups at **/user/lib/group**. To workaround this, you must extract the groups and users information of the existing RHEL for Edge system and fixating them as part of the Containerfile. Configure the Containerfile to copy a local **lib/group** file to the container image.

You can manually change the permissions of private keys, the **/etc** folder is mutable in edge systems. However, it does not solve the problem, because after running the **bootc switch** command, the Image Mode based system has the **ssh_keys** group configured with the ID 999. This value comes from the RHEL bootc base image, and this drift from ID 101 to ID 999 left the edge system unreachable through SSH. To fix this issue, follow the steps:

Prerequisites

- An existing RHEL for Edge **rpm-ostree** based system.
- 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.

Procedure

1. Extract the information of users and groups from the RHEL for Edge system.

```
$ mkdir -p ./usr/lib
$ ssh admin@192.168.100.50 'cat /lib/passwd' > ./usr/lib/passwd
$ ssh admin@192.168.100.50 'cat /lib/group' > ./usr/lib/group
```

2. Include the missing RHEL for Edge packages in the bootc based system by specifying them in a Containerfile. Additionally, use the COPY command to include the **group** and **passwd** content that was extracted from the RHEL for Edge system. The following is an example:

```
FROM registry.redhat.io/rhel9/rhel-bootc
WORKDIR /tmp
RUN dnf -y install ModemManager \
        NetworkManager-wifi \
        NetworkManager-wwan \
        audit \
        checkpolicy \
        clevis \
        clevis-dracut \
        clevis-luks \
        clevis-pin-tpm2 \
        clevis-systemd \
        containernetworking-plugins \
        dnsmasq \
        dracut-config-generic \
        fdo-client \
        fdo-owner-cli \
        firewalld \
        firewalld-filesystem \
        greenboot \
        greenboot-default-health-checks \
        grubby \
        ignition \
        ignition-edge \
        ipset \
        iwl100-firmware \
        iwl1000-firmware \
        iwl105-firmware \
        iwl135-firmware \
        iwl2000-firmware \
        iwl2030-firmware \
        iwl3160-firmware \
        iwl5000-firmware \
        iwl5150-firmware \
        iwl6050-firmware \
        iwl7260-firmware \
        libsecret \
        pinentry \
        policycoreutils-python-utils \
        python3-distro \
        python3-setools \
        rsync \
        setools-console \
        tmux \
        traceroute \
        usbguard \
        usbguard-selinux \
```

```
            wireless-regdb \
            wpa_supplicant

COPY etc /etc
# You can find the passwd and group content that were extracted from the RHEL for Edge system usr/lib/ in your current working directory. You can copy the content into the container image with the following step:
COPY usr /usr
```

3. Build the bootc image and push it to the registry:

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

4. Run the **bootc switch** command to the newly created bootable container image.

```
$ ssh admin@192.168.100.50
$ sudo bootc switch quay.io/<namespace>/<image>:<tag>
$ sudo reboot
```

Verification

After rebooting the edge system into the bootable container image, confirm that the contents of **/lib/passwd** and **/lib/group** match the content that was extracted from the OSTree system.

1. Check the content of **/lib/passwd**.

```
$ cat /lib/passwd
```

2. Check the content of **/lib/group**.

```
$ cat /lib/group
```

2.6.2. Switching a 9.6 RHEL for Edge installed by using a raw image to image mode

Use an existing 9.6 RHEL for Edge that you installed by using a raw image to switch to image mode for RHEL .

Prerequisites

- An existing 9.6 RHEL for Edge installed with a raw image.

Procedure

1. Update your image. See Updating RHEL for Edge images .
2. Switch your existing image from RHEL image builder to image mode.
   a. Build an image from rhel-bootc. For example:

```
$ cat Containerfile
FROM registry.redhat.io/rhel9/rhel-bootc:latest
RUN dnf install -y  \
    clevis  \
```

```
    clevis-dracut \
    clevis-luks \
    fdo-client \
    fdo-owner-cli
```

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

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

   a. Push the image to a registry

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

   b. Run **bootc switch** on the device.

```
$ bootc switch quay.io/<namespace>/<image>:_<tag>_
```

   c. Run **systemctl reboot**.

```
$ sudo systemctl reboot
```

Verification

- Connect to your RHEL for Edge system and use **bootc status**:

```
# bootc status
```

2.6.3. Switching a 9.6 RHEL for Edge installed by using a simplified-installer image to image mode

Use an existing 9.6 RHEL for Edge that you installed by using a **simplified-installer** image.

Prerequisites

- An existing 9.6 RHEL for Edge installed with a **simplified-installer** image.

Procedure

1. Check if **bootc** is installed:

```
$ rpm -qa | bootc
```

   Update your image to the latest **rpm-ostree** installation. See Updating RHEL for Edge images.

   a. Build an image from **rhel-bootc**. For example:

```
$ cat Containerfile
FROM registry.redhat.io/rhel9/rhel-bootc:latest
RUN dnf install -y \
    clevis \
    clevis-dracut \
```

```
    clevis-luks \
    fdo-client \
    fdo-owner-cli
```

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

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

   a. Push the image to a registry.

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

   b. Run **bootc switch** to switch the device to the image you pushed to the registry.

```
$ bootc switch quay.io/<namespace>/<image>:_<tag>_
```

   c. Run systemctl reboot.

```
$ sudo systemctl reboot
```

3. Verification

- Connect to your RHEL for Edge system and use **bootc status**:

```
# bootc status
```

2.7. UPGRADING FROM AN EXISTING RPM-OSTREE INSTALLATION TO IMAGE MODE FOR RHEL 10.0

You can upgrade an existing RHEL for Edge system based on RHEL 9.6 to use image mode for RHEL 10.0 by following these steps:

2.7.1. Upgrading an existing RHEL for Edge system to image mode for RHEL 10.0

You can upgrade an existing RHEL for Edge 9.6 system to RHEL 10.0 by using image mode for RHEL.

Prerequisites

- An existing 9.6 RHEL for Edge system.

Procedure

1. Update your image. See Updating RHEL for Edge images.

```
$ sudo rpm-ostree upgrade
$ sudo systemctl reboot
```

2. Build a bootc image that uses RHEL 10.0. For example:

```
$ cat Containerfile
FROM registry.redhat.io/rhel10/rhel-bootc:10.0
RUN dnf install -y \
```

```
clevis \
clevis-dracut \
clevis-luks \
fdo-client \
fdo-owner-cli
```

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

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

4. Push the image to a registry

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

5. Run **bootc switch** on the device.

```
$ bootc switch quay.io/<namespace>/<image>:_<tag>_
```

   a. Reboot the system.

```
$ sudo systemctl reboot
```

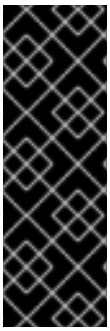
Verification

- Connect to your RHEL for Edge system and use **bootc status**:

```
# bootc status
```

CHAPTER 3. AUTOMATICALLY PROVISIONING AND ONBOARDING RHEL FOR EDGE DEVICES WITH FDO

Use image mode for RHEL to build operating system images suitable for your edge deployments. The FIDO Device Onboarding (FDO) process automatically provisions and onboards your Edge devices, and exchanges data with other devices and systems connected on the networks.



IMPORTANT

Red Hat provides the **FDO** process as a Technology Preview feature and should run on secure networks. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

3.1. THE FIDO DEVICE ONBOARDING (FDO) PROCESS

The FIDO Device Onboarding (FDO) is the process that:

- Provisions and onboards a device.
- Automatically configures credentials for this device. The FDO process is an automatic onboarding mechanism that is triggered by the installation of a new device.
- Enables this device to securely connect and interact on the network.

With FIDO Device Onboarding (FDO), you can perform a secure device onboarding by adding new devices into your IoT architecture. This includes the specified device configuration that needs to be trusted and integrated with the rest of the running systems. The FDO process is an automatic onboarding mechanism that is triggered by the installation of a new device.

The FDO protocol performs the following tasks:

- Solves the trust and chain of ownership along with the automation needed to securely onboard a device at scale.
- Performs device initialization at the manufacturing stage and late device binding for its actual use. This means that actual binding of the device to a management system happens on the first boot of the device without requiring manual configuration on the device.
- Supports automated secure devices onboarding, that is, zero touch installation and onboarding that does not need any specialized person at the edge location. After the device is onboarded, the management platform can connect to it and apply patches, updates, and rollbacks.

With FDO, you can benefit from the following:

- FDO is a secure and simple way to enroll a device to a management platform. Instead of embedding a Kickstart configuration to the image, FDO applies the device credentials during the device first boot directly to the ISO image.
- FDO solves the issue of late binding to a device, enabling any sensitive data to be shared over a secure FDO channel.

- FDO cryptographically identifies the system identity and ownership before enrolling and passing the configuration and other secrets to the system. That enables non-technical users to power-on the system.

The FDO protocol is based on the following servers:

Manufacturing server

1. Generates the device credentials.
2. Creates an Ownership voucher that is used to set the ownership of the device, later in the process.
3. Binds the device to a specific management platform.

Owner management system

1. Receives the Ownership voucher from the Manufacturing server and becomes the owner of the associated device.
2. Later in the process, it creates a secure channel between the device and the Owner onboarding server after the device authentication.
3. Uses the secure channel to send the required information, such as files and scripts for the onboarding automation to the device.

Service-info API server

Based on Service-info API server's configuration and modules available on the client, it performs the final steps of onboarding on target client devices, such as copying SSH keys and files, executing commands, creating users, encrypting disks and so on

Rendezvous server

1. Gets the Ownership voucher from the Owner management system and makes a mapping of the device UUID to the Owner server IP. Then, the Rendezvous server matches the device UUID with a target platform and informs the device about which Owner onboarding server endpoint this device must use.
2. During the first boot, the Rendezvous server will be the contact point for the device and it will direct the device to the owner, so that the device and the owner can establish a secure channel.

Device client

This is installed on the device. The Device client performs the following actions:

1. Starts the queries to the multiple servers where the onboarding automation will be executed.
2. Uses TCP/IP protocols to communicate with the servers.

At the **Device Initialization**, the device contacts the Manufacturing server to get the FDO credentials, a set of certificates and keys to be installed on the operating system with the Rendezvous server endpoint (URL). It also gets the Ownership Voucher, that is maintained separately in case you need to change the owner assignment.

1. The Device contacts the Manufacturing server

2. The Manufacturing server generates an Ownership Voucher and the Device Credentials for the Device.

3. The Ownership Voucher is transferred to the Owner onboarding server.

At the **On-site onboarding**, the Device gets the Rendezvous server endpoint (URL) from its device credentials and contacts Rendezvous server endpoint to start the onboarding process, which will redirect it to the Owner management system, that is formed by the Owner onboarding server and the Service Info API server.

4. The Owner onboarding server transfers the Ownership Voucher to the Rendezvous server, which makes a mapping of the Ownership Voucher to the Owner.

5. The device client reads device credentials.

6. The device client connects to the network.

7. After connecting to the network, the Device client contacts the Rendezvous server.

8. The Rendezvous server sends the owner endpoint URL to the Device Client, and registers the device.

9. The Device client connects to the Owner onboarding server shared by the Rendezvous server.

10. The Device proves that it is the correct device by signing a statement with a device key.

11. The Owner onboarding server proves itself correct by signing a statement with the last key of the Owner Voucher.

12. The Owner onboarding server transfers the information of the Device to the Service Info API server.

13. The Service info API server sends the configuration for the Device.

14. The Device is onboarded.

3.2. FDO AUTOMATIC ONBOARDING TECHNOLOGIES

Following are the technologies used in context to FDO automatic onboarding.

Table 3.1. OSTree and rpm-ostree terminology

Technology	Definition
UEFI	Unified Extensible Firmware Interface.
RHEL	Red Hat® Enterprise Linux® operating system
rpm-ostree	Background image-based upgrades.
Greenboot	**Healthcheck** framework for systemd on **rpm-ostree**.

Technology	Definition
image mode for RHEL	The new deployment method that uses containers to build system for operating system artifacts.
Container	A Linux® container is a set of 1 or more processes that are isolated from the rest of the system.
Coreos-installer	Assists installation of RHEL images, boots systems with UEFI.
FIDO FDO	Specification protocol to provision configuration and onboarding devices.

3.3. AUTOMATICALLY PROVISIONING AND ONBOARDING RHEL FOR EDGE DEVICES

Build image with **podman build** and automatically onboard it. After you boot the image, it provisions a RHEL for Edge system that you can use on a hard disk or as a boot image in a virtual machine.

Automatically provisioning and onboarding a RHEL for Edge device involves the following high-level steps:

1. Install and register a RHEL system.
2. Create a Containerfile, for example:

```
$ cat Containerfile
FROM registry.redhat.io/rhel10/rhel-bootc:10
#fdo
RUN dnf install -y fdo-init fdo-client && \
systemctl enable fdo-client-linuxapp.service

RUN mkdir -p /usr/lib/bootc/kargs.d && \ cat <<`EOF` >> /usr/lib/bootc/kargs.d/console.toml
kargs = ["inst.device=/dev/sda"] EOF
```

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

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

   At this point, the FDO server infrastructure should be up and running, and the specific onboarding details handled by the **service-info API** server, that is part of the owner infrastructure, are configured.

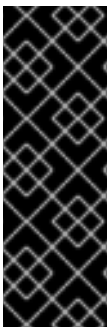
4. Install the image to a device. The FDO client runs on the image and the UEFI directory structure makes the image bootable.
5. The network configuration enables the device to reach out to the manufacturing server to perform the initial device credential exchange.
6. After the system reaches the endpoint, the device credentials are created for the device.

7. The device uses the device credentials to reach the Rendezvous server, where it checks the cryptographic credentials based on the vouchers that the Rendezvous server has, and then the Rendezvous server redirects the device to the Owner server.

8. The device contacts the Owner server. They establish a mutual trust and the final steps of onboarding happen based on the configuration of the Service-info API server. For example, it installs the SSH keys in the device, transfer the files, create the users, run the commands, encrypt the filesystem, and so on.

3.4. GENERATING KEY AND CERTIFICATES

To run the FIDO Device Onboarding (FDO) infrastructure, you need to generate keys and certificates. FDO generates these keys and certificates to configure the manufacturing server. FDO automatically generates the certificates and **.yaml** configuration files when you install the services, and re-creating them is optional. After you install and start the services, it runs with the default settings.



IMPORTANT

Red Hat provides the **fdo-admin-tool** tool as a Technology Preview feature and should run on secure networks. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

Prerequisites

- You installed the **fdo-admin-cli** RPM package

Procedure

1. Generate the keys and certificates in the **/etc/fdo** directory:

```
$ for i in "diun" "manufacturer" "device-ca" "owner"; do fdo-admin-tool generate-key-and-cert $i; done
$ ls keys device_ca_cert.pem device_ca_key.der diun_cert.pem diun_key.der manufacturer_cert.pem manufacturer_key.der owner_cert.pem owner_key.der
```

2. Check the key and certificates that were created in the **/etc/fdo/keys** directory:

```
$ tree keys
```

You can see the following output:

```
- device_ca_cert.pem
- device_ca_key.der
- diun_cert.pem
- diun_key.dre
- manufacturer_cert.pem
- manufacturer_key.der
- owner_cert.pem
- owner_key.pem
```

Additional resources

- See the **fdo-admin-tool generate-key-and-cert --help** command output

3.5. INSTALLING AND RUNNING THE MANUFACTURING SERVER

The **fdo-manufacturing-server** RPM package enables you to run the Manufacturing Server component of the FDO protocol. It also stores other components, such as the owner vouchers, the manufacturer keys, and information about the manufacturing sessions. During the device installation, the Manufacturing server generates the device credentials for the specific device, including its GUID, rendezvous information and other metadata. Later on in the process, the device uses this rendezvous information to contact the Rendezvous server.

IMPORTANT

Red Hat provides the **fdo-manufacturing-server** tool as a Technology Preview feature and should run on secure networks because Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

To install the **manufacturing server** RPM package, complete the following steps:

Procedure

1. Install the **fdo-admin-cli** package:

```
# dnf install -y fdo-admin-cli
```

2. Check if the **fdo-manufacturing-server** RPM package is installed:

```
$ rpm -qa | grep fdo-manufacturing-server --refresh
```

3. Check if the files were correctly installed:

```
$ *ls /usr/share/doc/fdo*SSSS
```

You can see the following output:

```
Output:
manufacturing-server.yml
owner-onboarding-server.yml
rendezvous-info.yml
rendezvous-server.yml
serviceinfo-api-server.yml
```

4. Optional: Check the content of each file, for example:

```
$ cat /usr/share/doc/fdo/manufacturing-server.yml
```

5. Configure the Manufacturing server. You must provide the following information:

   - The Manufacturing server URL
   - The IP address or DNS name for the Rendezvous server
   - The path to the keys and certificates that you generated.
     You can find an example of a Manufacturing server configuration file in the **/usr/share/doc/fdo/manufacturing-server.yml** directory. The following is a **manufacturing server.yml** example that is created and saved in the **/etc/fdo** directory. It contains paths to the directories, certificates, keys that you created, the Rendezvous server IP address and the default port.

```
session_store_driver:
  Directory:
    path: /etc/fdo/stores/manufacturing_sessions/
ownership_voucher_store_driver:
  Directory:
    path: /etc/fdo/stores/owner_vouchers
public_key_store_driver:
  Directory:
    path: /etc/fdo/stores/manufacturer_keys
bind: "0.0.0.0:8080"
protocols:
  plain_di: false
  diun:
    mfg_string_type: SerialNumber
    key_type: SECP384R1
    allowed_key_storage_types:
      - Tpm
      - FileSystem
    key_path: /etc/fdo/keys/diun_key.der
    cert_path: /etc/fdo/keys/diun_cert.pem
rendezvous_info:
  - deviceport: 8082
    ip_address: 192.168.122.99
    ownerport: 8082
    protocol: http
manufacturing:
  manufacturer_cert_path: /etc/fdo/keys/manufacturer_cert.pem
  device_cert_ca_private_key: /etc/fdo/keys/device_ca_key.der
  device_cert_ca_chain: /etc/fdo/keys/device_ca_cert.pem
  owner_cert_path: /etc/fdo/keys/owner_cert.pem
  manufacturer_private_key: /etc/fdo/keys/manufacturer_key.der
```

6. Start the Manufacturing server.

   a. Check if the systemd unit file are in the server:

```
# systemctl list-unit-files | grep fdo | grep manufacturing fdo-manufacturing-server.service
disabled disabled
```

   b. Enable and start the manufacturing server.

```
# systemctl enable --now fdo-manufacturing-server.service
```

c. Open the default ports in your firewall:

```
# firewall-cmd --add-port=8080/tcp --permanent
# systemctl restart firewalld
```

d. Ensure that the service is listening on the port 8080:

```
# ss -ltn
```

7. Install RHEL for Edge onto your system using the simplified installer.

Additional resources

- The manufacturing-server.yml example
- FDO automatic onboarding terminology

3.6. INSTALLING, CONFIGURING, AND RUNNING THE RENDEZVOUS SERVER

Install the **fdo-rendezvous-server** RPM package to enable the systems to receive the voucher generated by the Manufacturing server during the first device boot. The Rendezvous server then matches the device UUID with the target platform or cloud and informs the device about which Owner server endpoint the device must use.

Prerequisites

- You created a **manufacturer_cert.pem** certificate.
- You copied the **manufacturer_cert.pem** certificate to the **/etc/fdo/keys** directory in the Rendezvous server.

Procedure

1. Install the **fdo-rendezvous-server** RPM packages:

```
# dnf install -y fdo-rendezvous-server
```

2. Create the **rendezvous-server.yml** configuration file, including the path to the manufacturer certificate. You can find an example in **/usr/share/doc/fdo/rendezvous-server.yml**. The following example shows a configuration file that is saved in **/etc/fdo/rendezvous-server.yml**.

```
storage_driver:
  Directory:
    path: /etc/fdo/stores/rendezvous_registered
session_store_driver:
  Directory:
    path: /etc/fdo/stores/rendezvous_sessions
trusted_manufacturer_keys_path: /etc/fdo/keys/manufacturer_cert.pem
max_wait_seconds: ~
bind: "0.0.0.0:8082"
```

3. Check the Rendezvous server service status:

```
# systemctl list-unit-files | grep fdo | grep rende
fdo-rendezvous-server.service disabled disabled
```

a. If the service is stopped and disabled, enable and start it:

```
# systemctl enable --now fdo-rendezvous-server.service
```

4. Check that the server is listening on the default configured port 8082:

```
# ss -ltn
```

5. Open the port if you have a firewall configured on this server:

```
# firewall-cmd --add-port=8082/tcp --permanent
# systemctl restart firewalld
```

3.7. INSTALLING, CONFIGURING, AND RUNNING THE OWNER SERVER

Install the **fdo-owner-cli** and **fdo-owner-onboarding-server** RPM package to enable the systems to receive the voucher generated by the Manufacturing server during the first device boot. The Rendezvous server then matches the device UUID with the target platform or cloud and informs the device about which Owner server endpoint the device must use.

Prerequisites

- The device where the server will be deployed has a Trusted Platform Module (TPM) device to encrypt the disk. If not, you will get an error when booting the RHEL for Edge device.
- You created the **device_ca_cert.pem**, **owner_key.der**, and **owner_cert.pem** with keys and certificates and copied them into the **/etc/fdo/keys** directory.

Procedure

1. Install the required RPMs in this server:

```
# dnf install -y fdo-owner-cli fdo-owner-onboarding-server
```

2. Prepare the **owner-onboarding-server.yml** configuration file and save it to the **/etc/fdo/** directory. Include the path to the certificates you already copied and information about where to publish the Owner server service in this file.
The following is an example available in **/usr/share/doc/fdo/owner-onboarding-server.yml**. You can find references to the Service Info API, such as the URL or the authentication token.

```
---
ownership_voucher_store_driver:
  Directory:
    path: /etc/fdo/stores/owner_vouchers
session_store_driver:
  Directory:
    path: /etc/fdo/stores/owner_onboarding_sessions
trusted_device_keys_path: /etc/fdo/keys/device_ca_cert.pem
owner_private_key_path: /etc/fdo/keys/owner_key.der
owner_public_key_path: /etc/fdo/keys/owner_cert.pem
```

```
bind: "0.0.0.0:8081"
service_info_api_url: "http://localhost:8083/device_info"
service_info_api_authentication:
  BearerToken:
    token: Kpt5P/5flBkaiNSvDYS3cEdBQXJn2Zv9n1D50431/lo=
owner_addresses:
  - transport: http
    addresses:
      - ip_address: 192.168.122.149
```

3. Create and configure the Service Info API.

   a. Add the automated information for onboarding, such as user creation, files to be copied or created, commands to be executed, disk to be encrypted, and so on. Use the Service Info API configuration file example in **/usr/share/doc/fdo/serviceinfo-api-server.yml** as a template to create the configuration file under **/etc/fdo/**.

```
---
service_info:
  initial_user:
    username: admin
    sshkeys:
    - "ssh-rsa AAAA...."
  files:
  - path: /root/resolv.conf
    source_path: /etc/resolv.conf
  commands:
  - command: touch
    args:
    - /root/test
    return_stdout: true
    return_stderr: true
  diskencryption_clevis:
  - disk_label: /dev/vda4
    binding:
      pin: tpm2
      config: "{}"
    reencrypt: true
  additional_serviceinfo: ~
bind: "0.0.0.0:8083"
device_specific_store_driver:
  Directory:
    path: /etc/fdo/stores/serviceinfo_api_devices
service_info_auth_token: Kpt5P/5flBkaiNSvDYS3cEdBQXJn2Zv9n1D50431/lo=
admin_auth_token: zJNoErq7aa0RusJ1w0tkTjdITdMCWYkndzVv7F0V42Q=
```

4. Check the status of the systemd units:

```
# systemctl list-unit-files | grep fdo
fdo-owner-onboarding-server.service        disabled        disabled
fdo-serviceinfo-api-server.service         disabled        disabled
```

   a. If the service is stopped and disabled, enable and start it:

```
# systemctl enable --now fdo-owner-onboarding-server.service
# systemctl enable --now fdo-serviceinfo-api-server.service
```

NOTE

You must restart the **systemd** services every time you change the configuration files.

5. Check that the server is listening on the default configured port 8083:

```
# ss -ltn
```

6. Open the port if you have a firewall configured on this server:

```
# firewall-cmd --add-port=8081/tcp --permanent
# firewall-cmd --add-port=8083/tcp --permanent
# systemctl restart firewalld
```

3.8. AUTOMATICALLY ONBOARDING A RHEL FOR EDGE DEVICE BY USING FDO AUTHENTICATION

To prepare your device to automatically onboard a RHEL for Edge device and provision it as part of the installation process, complete the following steps:

Prerequisites

- You built a customized images by using **podman build**.
- Your device is assembled.
- You installed the **fdo-manufacturing-server** RPM package. See Installing the manufacturing server package.

Procedure

1. Start the installation process by booting the RHEL for Edge image on your device. You can install it from a CD-ROM or from a USB flash drive, for example.
2. Verify through the terminal that the device has reached the manufacturing service to perform the initial device credential exchange and has produced an ownership voucher.
   You can find the ownership voucher at the storage location configured by the ***ownership_voucher_store_driver*:** parameter at the ***manufacturing-sever.yml*** file.

   The directory should have an ***ownership_voucher*** file with a name in the GUID format which indicates that the correct device credentials were added to the device.

   The onboarding server uses the device credential to authenticate against the onboarding server. It then passes the configuration to the device. After the device receives the configuration from the onboarding server, it receives an SSH key and installs the operating system on the device. Finally, the system automatically reboots, encrypts it with a strong key stored at TPM.

Verification

After the device automatically reboots, you can log in to the device with the credentials that you created as part of the FDO process.

- Log in to the device by providing the username and password you created in the Service Info API.

3.9. DEPLOYING AN IMAGE MODE FOR RHEL SYSTEMS BY USING FDO

You can deploy an image mode for a RHEL system by using FIDO Device Onboarding (FDO) to deliver configuration to this system. Use a Kickstart file to configure various parts of the installation process, such as setting up users, customizing partitioning, and adding an SSH key. You can include the Kickstart file in an ISO build to configure any part of the installation process, except the deployment of the base image.

If you use an ISO with a bootc container base image, **bootc-image-builder automatically** installs **ostreecontainer**, the command to install the container image. You can still configure anything, except the **ostreecontainer** command.

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.
- You have FDO server infrastructure deployed.

Procedure

1. Create a Containerfile, for example:

```
FROM registry.redhat.io/rhel10/rhel-bootc:latest
RUN dnf install -y fdo-init fdo-client
RUN systemctl enable fdo-client-linuxapp.service
```

2. Create your Kickstart file. The following Kickstart file is an example of a fully unattended Kickstart file configuration that contains user creation and partition instructions.

```
[customizations.installer.kickstart]
contents = """
text --non-interactive
zerombr
clearpart --all --initlabel --disklabel=gpt
autopart --noswap --type=lvm
user --name=test --groups=wheel --plaintext --password=test
sshkey --username=test "ssh-ed25519 AAA..."
network --bootproto=dhcp --device=link --activate --onboot=on
poweroff

%post
export MANUFACTURING_SERVER_URL="http://192.168……"
export DIUN_PUB_KEY_INSECURE="true"
/usr/libexec/fdo/fdo-manufacturing-client
```

```
%end

"""
```

In the export *<MANUFACTURING_SERVER_URL>* field, replace the manufacturing server URL with your own manufacturing server URL.

3. Save the Kickstart configuration in the **.toml** format to inject the Kickstart content. For example, **config.toml**.

4. Create the following folder:

```
$ mkdir $(pwd)/output"
```

5. Run **bootc-image-builder**, and include the Kickstart file configuration that you want to add to the ISO build. The **bootc-image-builder** automatically adds the **ostreecontainer** command that installs the container image.

```
$ sudo podman run \
    --rm \
    -it \
    --privileged \
    --pull=newer \
    --security-opt label=type:unconfined_t \
    -v /var/lib/containers/storage:/var/lib/containers/storage \
    -v $(pwd)/config.toml:/config.toml \
    -v $(pwd)/output:/output \
    registry.redhat.io/rhel10/bootc-image-builder:latest \
    --type iso \
    --config /config.toml \
  quay.io/<namespace>/<image>:<tag>
```

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

Additional resources

- The Automatically provisioning and onboarding RHEL for Edge devices with FDO documentation