2025 - RH1 RHEL Image Building

Travis Michette tmichett@redhat.com

Table of Contents

This workshop uses blocks which contain runnable commands and represent output. The blocks marked with `bash`, `dockerfile`, etc in the upper right corner can be copied and pasted into a terminal window instead of being typed. This is provided as a convenience.

Chapter 1. System setup

In this lab, you will be cloning the lab helper scripts and preparing the host, as well as setting up the required access for Red Hat resources.

1.1. Installing Repositories

First, let's start by cloning the helper scripts.

```
git clone https://github.com/nzwulfin/rh-summit-2024-lb1506
```

...and changing our directory to the newly created one:

```
cd rh-summit-2024-lb1506
```

1.2. Red Hat registry login

In order to use the base Red Hat Enterprise Linux container images, you need to login to Red Hat's image container registry by using your Red Hat account credentials. If you don't have a Red Hat account or are concerned about using your credentials on a public network, feel free to create a new personal account that povides you with a no-cost Red Hat Developer Subscription.

```
podman login registry.redhat.io
```

You will be asked for a username (this is typically the email address associated with your account) and a password.

1.3. Preparing the host for container building

In this final part of the setup, you will:

- setup the hypervisor for running virtual machines
- generate a local ssh key
- run a local container image registry
- download all the required container images

Don't worry if this sounds like a lot, there is a script to do it all for you:

```
make setup
```

1.4. Checking the setup

One final check before you advance to the next lab.

```
make status
```

The output from the command above should be similar to the following:

```
git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
podman login --get-login registry.redhat.io
cgament@redhat.com
podman image exists "registry.redhat.io/rhel9/rhel-bootc:9.4"
podman image exists "quay.io/centos-bootc/bootc-image-builder:latest"
podman image exists registry.access.redhat.com/ubi9/ubi-minimal
podman image exists registry.access.redhat.com/ubi9/ubi
podman image exists quay.io/kwozyman/toolbox:httpd
podman image exists quay.io/kwozyman/toolbox:registry
     Active: active (running) since Wed 2024-05-01 17:13:37 EDT; 55s ago
 Id   Name   State
--------------------

net.ipv4.ip_unprivileged_port_start = 80
ID            NAME             CPU %       MEM USAGE / LIMIT  MEM %       NET IO
BLOCK IO    PIDS        CPU TIME    AVG CPU %
fa77d569b19f  summit-registry  0.05%       17.9MB / 67.3GB    0.03%       3.88kB /
1.888kB  0B / 0B     20          18.923277s  0.05%
```

Chapter 2. Build a bootc container image

In this step you will transform an application container to a bootc image.

2.1. Modify the Containerfile

The `vim` editor is also available. Users of `vim` should already know the basics of opening, saving and exiting the editor. The instructions will reflect `nano` commands. All of the files used can be found in the `examples` directory.

Let's start with a typical application container and convert it to an image mode host. We'll start with something simple, a basic webserver running on the Red Hat Universal Base Image. To start the editor:

```
nano Containerfile
```

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

RUN dnf install -y httpd
RUN echo "Hello Red Hat" > /var/www/html/index.html

ENTRYPOINT /usr/sbin/httpd -DFOREGROUND
```

A brief explanation of the key directives in the Containerfile above:

- `FROM` → the base container image
- `ENTRYPOINT` → the command that will be executed by default at run time

You can now replace the contents as follows:

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

COPY certs/004-summit.conf /etc/containers/registries.conf.d/004-summit.conf

RUN dnf install -y httpd
RUN echo "Hello Red Hat" > /var/www/html/index.html

RUN systemctl enable httpd.service
```

In `nano`, you can save and exit with `Ctrl+x`, followed by the key `y` at the "Save modified buffer?" prompt and the `Enter` key at the prompt to keep the filename.

Let's talk about the effects of these changes:

- The `FROM` field is now pointing to the Red Hat Enterprise Linux 9.4 bootc image instead of the UBI base image. This new bootc image is a complete version of RHEL 9.4, much like the KVM image available on the Customer Portal. This image includes all of the OS components to run as a host as well as the new `bootc` commands. It is built using `ostree` to facilitate the transactional updates that marks an image mode deployment from a package mode host.
- `RUN systemctl enable httpd.service` replaces the `ENTRYPOINT`. To start the `httpd` service at host startup as you use `systemctl` would with any other host. There's no need to specify an executable that should be run when the container is started by a container engine like podman.
- `COPY certs/004-summit.conf⋯` copies a single file from our helper directory into the container. This config snippet will allow us to pull container images without TLS verification in the lab environment. This is only necessary based on the use of self-signed certificates with the lack of an internal CA available.

2.2. Build the first bootc image

As with application containers, you can build this container with standard tools:

- `--file Containerfile` → what Containerfile to use for the build
- `--tag summit.registry/lb1506:bootc` → the "tag" or name of the resulting container in the format <registry>/<repository>:<tag>

```
podman build --file Containerfile --tag summit.registry/lb1506:bootc
```

If the above command is succesful, the last two lines of the output should read like this:

```
Successfully tagged summit.registry/lb1506:bootc
f1bea10eb37acf2e78a9b01c6242110c1901adbaf40dbce479241c6c735c58da
```

2.3. Distributing the bootc image

Now that we have a full operating system in a standard OCI container image, we can use standard container registries to host and distribute our OS images. The only atypical thing so far is what's in the base image. How we added software or files to the image is the same as any other application container. The special sauce of `bootc` comes when a host is running.

We have a registry available within the lab environment we can push our image to:

```
podman push summit.registry/lb1506:bootc
```

That’s all there is to designing and building a complete operating system. Start with a base, then add software and related configurations. You not only have the record of that work you can put under source control in the form of the Containerfile, you also have the binary image stored and versioned in the registry.

You’ve built your first bootc image, and it’s now published for use. How do we get from here to a running system?

Chapter 3. Deploy a virtual machine with bootc image

In this lab, you will build a `qcow2` disk image from the bootc image and then launch a virtual machine from it. `qcow2` is a standard file format used by the Linux virtualization system.

This exercise includes both a make command and the commands executed by that make target. You do not need to execute both.

3.1. Create the config for building the virtual machine image

To deploy our bootc image as a host, we need to get the contents of the image onto a disk. While `bootc` handles those mechanics, the actual creation of a phyiscal disk, virtual disk, or cloud image are handled by other standard tools. For example, Anaconda can be used with bootc images for phyiscal or virtual hosts like you would today, but without the `%packages` block we use today. In this lab, we'll use the `bootc-image-builder` tool which can create various different disk image types and call `bootc` to deploy the image contents.

You may have noticed the bootc image we've created does not include any login credentials. Not a typical concern for an application container, but as a host we will very likely need to interact at some point.

Since we are defining an image to be shared by multiple hosts in multiple different environments, users and authentication is likely to be handled at the deployment stage, not the build stage.

There are cases where it may be useful for a user and credentials to be added to an image, as a 'break glass' emergency login for example.

To add a user during the conversion to a disk image, you need to create a JSON file with the credentials. This config file will be used by `bootc-image-builder` to customize the final disk image.

A ssh key has been generated in the Setup section of this lab. You can view the public part like this:

```
cat ~/.ssh/id_rsa.pub
```

The output should look something like this (Example):

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAuXnpoluye+KM+9tvIAdHf+F0IHh+K73tlcjEG8LJRB lab-
user@hypervisor
```

You can now create a file called `config-qcow2.json` with the following contents, replacing "SSHKEY" in the `key` field with the contents of `~/.ssh/id_rsa.pub` from your system:

```
nano config-qcow2.json
```

```
{
  "blueprint": {
    "customizations": {
      "user": [
        {
          "name": "lab-user",
          "password": "lb1506",
          "key": "SSHKEY",
          "groups": [
            "wheel"
          ]
        }
      ]
    }
  }
}
```

This configuration blueprint creates a user in the resulting virtual machine with the login `lab-user`, the password `lb1506` and the ssh key of the user on the lab host.

For your convenience, a config file has already been generated with the ssh key for your lab host in `config/config-qcow2.json` so you can just use that with `cp config/config-qcow2.json config-qcow2.json`.

3.2. Create the disk image

First, you want to make sure the bootc image is available to the local registry:

```
skopeo inspect docker://summit.registry/lb1506:bootc | jq '.Digest'
```

```
"sha256:655721ff613ee766a4126cb5e0d5ae81598e1b0c3bcf7017c36c4d72cb092fe9"
```

To convert from an OCI image to a disk image, we have a special version of image builder that has support for `bootc`. This `bootc-image-builder` itself runs as a container, and as a result needs additional capabilities and to be run as `root`.

Try to generate the image now:

This could take 2+ minutes to complete.

```
sudo podman run --rm --privileged \
        --volume .:/output \
        --volume ./config-qcow2.json:/config.json \
        registry.redhat.io/rhel9/bootc-image-builder:latest \
        --type qcow2 --config /config.json \
        --tls-verify=false \
        summit.registry/lb1506:bootc
```

A brief explanation of the arguments used for `podman` and `bootc-image-buider`:

- `--rm` → do not keep the build container after execution finishes
- `--privileged` → add capabilities required to build the image
- `--volume` → podman will map these local directories or files to the container
- `registry.redhat.io/rhel9/bootc-image-builder:latest` → the image builder container image

From here, these arguments passed to `bootc-image-builder`

- `--type qcow2` → the type of image to build
- `--config /config.json` → the json configuration file used; please note this is relative to the container volume we added
- `--tls-verify=false` → the registry we are running is local and it has self signed certificates; this flag would not be needed with `quay.io` for example
- `summit.registry/lb1506:bootc` → the bootc image we are unpacking into the qcow2 disk

Once this completes, you will find the image is the `qcow2/` directory:

For your convenience, this part of the exercise is automated by using the `make qcow CONTAINER=summit.registry/lb1506:bootc` command.

```
ls -lah qcow2/disk.qcow2
```

The output should be similar to:

```
-rw-r--r--. 1 root root 945M May  2 06:10 qcow2/disk.qcow2
```

3.3. Create the virtual machine

The creation of virtual machines is out of scope for this lab, so we have provided a convenience command in the Makeile. We've detailed the commnds being run so you can follow along with what is happening. If you like, feel free to run the commands on your own instead of using the `make` target. The core of the `virt-install` command used is `--import` which skips the install process and creates the VM based on the provided disk image.

```
make vm-qcow
```

For reference, commands executed in the Makefile and displayed in the terminal are:

```
sudo cp qcow2/disk.qcow2 /var/lib/libvirt/images/summit/qcow-vm.qcow2
virt-install --connect qemu:///system \
                --name qcow \
                --disk /var/lib/libvirt/images/summit/qcow-vm.qcow2 \
                --import \
                --network "network=summit-network,mac=de:ad:be:ef:01:03" \
                --memory 4096 \
                --graphics none \
                --osinfo rhel9-unknown \
                --noautoconsole \
                --noreboot
virsh --connect qemu:///system start qcow
```

If `make vm-qcow` was successful, you should see the final line of output like this:

```
Domain 'qcow' started
```

Check to make sure the virtual machine running:

```
virsh --connect qemu:///system list
```

```
 Id   Name                State
------------------------------------
 1    qcow                running
```

3.4. Test and login to the virtual machine

Congratulations, you are running a bootc virtual machine! Now that the virtual machine is up and

running, you can see if the webserver behaves as expected.

```
curl http://qcow-vm
```

And the results should be the "Hello Red Hat" string defined in the index.html.

You can now login to the virtual machine.

```
ssh lab-user@qcow-vm
```

If the ssh key is not automatically picked up, use the password defined in the JSON file at the beginning of this lab (by default `lb1506`). This is also the password to use when prompted by `sudo`.

Once you have logged in, you can inspect the bootc status.

```
sudo bootc status
```

The output should look similar to this:

```
apiVersion: org.containers.bootc/v1alpha1
kind: BootcHost
metadata:
  name: host
spec:
  image:
    image: summit.registry/lb1506:bootc
    transport: registry
  bootOrder: default
status:
  staged: null
  booted: ①
    image:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240501.0
      timestamp: null
      imageDigest:
sha256:0a3daed6e31c2f2917e17ea994059e1aaee0481fe16836c118c5e1d10a87365c
    cachedUpdate: null
    incompatible: false
    pinned: false
```

```
    ostree:
      checksum: 008e3bef805f25224f591240627bea2a06ce12b25494836c2dab7d1b0a1691a8
      deploySerial: 0
  rollback: null
  rollbackQueued: false
  type: bootcHost
```

From the output of `bootc status`, find the block that starts with `booted`.

① This block provides information about the image in use. You can see that image is listed as `summit.registry/lb1506:bootc`.

You can explore the virtual machine before moving on to the next section:

- `systemctl status httpd` → see the `httpd` service we have enabled in the Containerfile
- `cat /var/www/html/index.html` → see the index.html file we have created in the Containerfile

Our services are running, but how can we tell that we are on system and not running a container? First, `bootc` can tell you directly if it's being run on an image mode host or not in the `bootc status` output. It will be all `null` values if run on a non-bootc enabled host.

For other ways, we can look at how the system was started and some of the characteristics that will change, like SELinux context.

Let's look at kernel command line as well as PID1 in the `/proc` filesystem and see what runtime info we have.

```
cat /proc/cmdline ①
cat /proc/1/cgroup ②
cat /proc/1/attr/current ③
```

We can see in the kernel command line some clear ties to an `ostree` partition, and our PID1 details shows systemd running with init scope from the cgroup hierarchy and SELinux context. We'll look at the container output in a later exercise, but the SELinux context would differ.

① `BOOT_IMAGE=(hd0,gpt3)/boot/ostree/default-6fe9dddacaf5c3232ba2332010aa7442e0a6d0e3f455b7572b047cc2284c3f2f/vmlinuz-5.14.0-427.26.1.el9_4.x86_64 root=UUID=5425bac2-bfc2-457d-93f8-ae7d3bf14d6d rw boot=UUID=9b9c7b0a-61c6-4a66-ade5-8c6690f1efa7 rw console=tty0 console=ttyS0 ostree=/ostree/boot.1/default/6fe9dddacaf5c3232ba2332010aa7442e0a6d0e3f455b7572b047cc2284c3f2f/0`

② `0::/init.scope`

③ `system_u:system_r:init_t:s0`

Before proceeding, make sure you have logged out of the virtual machine:

```
logout
```

The prompt should read `[lab-user@hypervisor rh-summit-2024-lb1506]$` before continuing.

Chapter 4. Updating bootc systems

In this lab, you will learn how to create updated images and apply them to bootc hosts.

4.1. Updating the bootc image

We've created our first system but there are a few things that might be missing we typically do to our systems. Not requiring a password every time we use sudo might be something we allow admins to do. We may also need to put notices on logins or present info in the MOTD. Let's make some changes to how the system behaves. While we're at it, lets update the index page too.

First, let's set up a directory structure that mimics `/etc` on our lab host and create the files we need inside it. This allows us to add all the configs at once with a single command and in a single layer of the image.

This is a local directory, relative to the location of the Containerfile, not the system `/etc/`. These files are also provided in the `examples/update` directory.

```
mkdir -p etc/sudoers.d
```

Since the motd file lives in the top level of `/etc`, that one command creates all the paths we need for this exercise. RHEL supports drop-in directories for several services and applications, including `sudo` privileges. Drop-in support lets us add a file to a `<servicename>.d` directory rather than editing the main configuration file. You might already be familiar with drop-in config files from working with Apache and it's `/etc/httpd/conf.d/` directory.

Let's address the sudoers password first, by allowing users in the wheel group to not need to enter their password.

```
nano etc/sudoers.d/wheel
```

Add the following and save the file

```
# Enable passwordless sudo for the wheel group
%wheel        ALL=(ALL)       NOPASSWD: ALL
```

And let's provide a message of the day!

```
nano etc/motd
```

Add the following and save the file

```
Welcome to the image mode lifestyle!
```

Now that we have our new configs, we can add them to the Containerfile and rebuild the image.

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

COPY certs/004-summit.conf /etc/containers/registries.conf.d/004-summit.conf

ADD etc/ /etc ①

RUN dnf install -y httpd
RUN echo "Hello Red Hat Summit Connect 2024!!" > /var/www/html/index.html

RUN systemctl enable httpd.service
```

① Adding the files we just created

If you aren't familiar, the `ADD` directive has some extra capabilities over the `COPY` command we used for the SSL certs. Where `COPY` only works on a single file, `ADD` can work on complete directories or even unpack tar files. This can be very useful when dealing with standard configs, but could make the Containerfile a little opaque if folks are unfamilar with the syntax.

We rebuild the image with our updates the same way we originally built it.

```
podman build --file Containerfile --tag summit.registry/lb1506:bootc
```

And make sure to push it to the registry:

```
podman push summit.registry/lb1506:bootc
```

4.2. Updating the virtual machine

The last virtual machine you have created during this workshop should still be running. You can check this with

```
virsh --connect qemu:///system list
```

And the output should the virtual machine called `qcow` as `running`.

Now you can ssh into the virtual machine:

```
ssh lab-user@qcow-vm
```

The `bootc update` command will download and prepare a detected update for use at the next boot. This means updates are offline, and can be applied at any time. Changes will only take effect when the system reboots. (Reminder, the password for `sudo` is `lb1506`):

```
sudo bootc update
```

The output should look something like this:

```
Loading usr/lib/ostree/prepare-root.conf
Queued for next boot: summit.registry/lb1506:bootc-auth
  Version: 9.20240501.0
  Digest: sha256:c5a5bc63cc5d081c528c82a177d0c5eac996a16fa3a651f93d07825302ff5336
Total new layers: 73    Size: 947.9 MB
Removed layers:   6     Size: 2.3 kB
Added layers:     6     Size: 2.2 kB
```

Notice that `bootc update` provides information about the layers that have been modified.

```
sudo bootc status
```

The output should look like:

```
apiVersion: org.containers.bootc/v1alpha1
kind: BootcHost
metadata:
  name: host
spec:
  image:
    image: summit.registry/lb1506:bootc
    transport: registry
  bootOrder: default
status:
  staged: ①
    image:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240501.0
```

```
      timestamp: null
      imageDigest:
sha256:c5a5bc63cc5d081c528c82a177d0c5eac996a16fa3a651f93d07825302ff5336
    cachedUpdate: null
    incompatible: false
    pinned: false
    ostree:
      checksum: 20cbee67379b96ad9eb273c0f7a7cd3673644e2d8af13f8b7437afd12dc95070
      deploySerial: 0
  booted:
    image:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240501.0
      timestamp: null
      imageDigest:
sha256:b57df8b24f7ddaf39ade0efe02d203e4fcd63deca2a9fd47f4af5c2cc3fcd017
    cachedUpdate:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240501.0
      timestamp: null
      imageDigest:
sha256:c5a5bc63cc5d081c528c82a177d0c5eac996a16fa3a651f93d07825302ff5336
    incompatible: false
    pinned: false
    ostree:
      checksum: 22b18bfa0e94fbe390379cb4bae150ebad85c8844e7b721179d26c1df636ce8e
      deploySerial: 0
  rollback: null
  rollbackQueued: false
  type: bootcHost
```

① You can see the staged changes in addition to current running state of the host in `bootc status`. The SHA in the `staged` block should match the Digest from the output of `bootc update`.

The last step for the change to take is to reboot the virtual machine. Before doing so, please make sure you are logged in to the virtual machine and not the hypervisor (the prompt should look like `[lab-user@qcow-vm ~]$`):

```
sudo systemctl reboot
```

4.3. Testing the changes

After a short wait, log back into the system and and you should see the message of the day after logging in successfully.

```
ssh lab-user@qcow-vm
```

We can check on our sudoers policy change as well. You shouldn't be prompted for you password:

```
sudo cat /etc/motd
```

What about the change to the index page?

```
curl http://localhost
sudo cat /var/www/html/index.html
```

```
Hello Red Hat
```

The new text doesn't appear, and it's also not in the file on disk. This is **expected** based on how `bootc` handles directories and image contents during changes.

Stay logged into the VM to explore this in the next module.

Chapter 5. Rolling back bootc systems

In this lab, you'll look at rollbacks and examine how bootc handles directories on updates.

5.1. Rolling back changes to the virtual machine

You should still be logged into the bootc VM (the prompt should look like `[lab@qcow-vm ~]$`), if not log back in now.

```
ssh lab-user@qcow-vm
```

We have a new built-in option available to image mode systems that typically takes more preparation with package mode operations: the rollback.

Since `bootc` manages state on disk, we have the previous working system available to us. Normally, we'd need to have set up snapshots or refer to a backup but `bootc` automatically provides us the rollback built in.

Let's check for an available rollback option to get us back to the previous image.

```
sudo bootc status
```

```
apiVersion: org.containers.bootc/v1alpha1
kind: BootcHost
metadata:
  name: host
spec:
  image:
    image: summit.registry/lb1506:bootc
    transport: registry
  bootOrder: default
status:
  staged: null ②
  booted: ①
    image:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240714.0
      timestamp: null
      imageDigest:
sha256:681f5f1b99985281f4fec2a0febc0ba1648aa6c91dc0f4f99c0ea1599edda8d7
    cachedUpdate: null
```

```
    incompatible: false
    pinned: false
    ostree:
      checksum: 3839bd1c5fa58d08beb28b55c275e4ca881c6fd866fb56856f525efa45f0ba6f
      deploySerial: 0
  rollback: ③
    image:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240714.0
      timestamp: null
      imageDigest:
sha256:9b49d2186c0e52142d0bc5ebace934420ba5108320c37ad985c423720fc69f29
    cachedUpdate:
      image:
        image: summit.registry/lb1506:bootc
        transport: registry
      version: 9.20240714.0
      timestamp: null
      imageDigest:
sha256:681f5f1b99985281f4fec2a0febc0ba1648aa6c91dc0f4f99c0ea1599edda8d7
    incompatible: false
    pinned: false
    ostree:
      checksum: 2d9ba2dbc14ea10d7c860bedafb19736e9d2f3a068f31f4545e5096a0369304e
      deploySerial: 0
  rollbackQueued: false ④
  type: bootcHost
```

Looking at the status output, we can see there's a section marked `rollback`. There are at most 3 images available on disk at any one time:

① the currently active image (`booted`)

② the image that will be actived next (`staged`)

③ the fallback image that we're be looking for here (`rollback`).

④ there's no currently queued rollback actions

Not all of these sections will have information at all times.

Rollbacks are as simple as running one command. Let's get this image back to the previous state then we can dig into what happened.

```
sudo bootc rollback
```

The command should return very fast and the output looks like this:

```
bootfs is sufficient for calculated new size: 0 bytes
Next boot: rollback deployment
```

```
sudo bootc status | grep Queued
```

You should see the value of `rollbackQueued` has been updated as well. This can be useful to check before restarting a sysytem.

```
  rollbackQueued: true
```

You can also check the boot order in the `spec` block to see what has been sent to the bootloader.

```
sudo bootc status | grep Order
```

```
  bootOrder: rollback
```

After the reboot, the `rollback` image will become the booted image, and will also become the new default in the boot order.

As usual, a reboot is needed to apply changes. Before doing so, please make sure you are logged in to the virtual machine and not the hypervisor (the prompt should look like `[lab@qcow-vm ~]$`):

```
sudo systemctl reboot
```

5.2. How image mode handles directories

On the lab host, we're going to make some changes to the Containerfile to account for how directories are managed by `bootc`.

In an image mode system, `bootc` manages available images on disk for updates and rollbacks. You just created an update, applied it, then returned to a previous version all through `bootc`. Behind the scenes, `bootc` handles the following directories differently, which is what allows for the seamless update and rollback experience.

- `/usr` → image state, contents of the image will be extracted and overwrite local files
- `/etc` → local configuration state, contents of the image are merged with a preference for local files

- `/var` → local state, contents of the image will be ignored after the inital installation of any image

The configuration files we added to `/etc` showed up after the update as a result of the merge treatment of bootc. If we'd made local changes that conflicted, we'd see the local changes rather than the new configs.

Our initial web page went in `/var` which means after it was unpacked from the original image, `bootc` treated it as local machine state. So the change in the Containerfile wasn't applied to the running host. Since we want to control everything about our webserver from the image, we'll need to make some changes to where we put content and how we serve it in Apache.

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

COPY certs/004-summit.conf /etc/containers/registries.conf.d/004-summit.conf

ADD etc/ /etc

RUN dnf install -y httpd

RUN <<EOF ①
    mv /var/www /usr/share/www
    sed -i 's-/var/www-/usr/share/www-' /etc/httpd/conf/httpd.conf
EOF

RUN echo "Hello Red Hat Summit Connect 2024!!" > /usr/share/www/html/index.html

RUN systemctl enable httpd.service
```

① Added line to run several commands using the heredoc format

Let's break down that new `RUN` directive.

The `httpd` package drops content in `/var/www` by default, and on bootc systems `/var` is machine local. For this example, we want to control web content in the image, we need to move it to somewhere under `bootc` control. In our Containerfile, we move the default package contents to a new location in `/usr` then update the Apache configuration to serve pages from this new directory. We've also changed the echo line to create the index.html in the new location.

Rebuild the image with our new configuration and index page. Since this is a rebuild, podman will reuse the existing layers if there are no changes. This makes updates faster and take less space. Notice the push to the registry also only pushes those layers that contain changes.

```
podman build --file Containerfile --tag summit.registry/lb1506:bootc
```

And make sure to push it to the registry:

```
podman push summit.registry/lb1506:bootc
```

5.3. Updating the virtual machine

Now you can ssh into the virtual machine

```
ssh lab-user@qcow-vm
```

Previously, we checked for an update, downloaded and staged it locally to be activated, then manually rebooted the system to have the update take effect. This is a very good procedure for a manual update or in places where we need to schedule any outages ahead of time, say during a maintenance window. We can do this all at once by adding a flag to the `update` command. This gives us a way to automate the process, like with a systemd timer. Image mode hosts ship with this timer by default.

```
systemctl list-timers bootc-fetch-apply-updates.timer
```

```
NEXT                    LEFT          LAST PASSED UNIT                    ACTIVATES
Wed 2024-07-24 16:13:⋯ 1h 44min left -    -      bootc-fetch-apply-upd⋯ bootc-fetch-
apply-upd⋯

1 timers listed.
Pass --all to see loaded but inactive timers, too.
```

Instead of waiting for this timer to trigger, we can immediately apply the new update and reboot.

```
sudo bootc update --apply
```

Remember that the update will detail what layers are new, removed, or added.

```
Loading usr/lib/ostree/prepare-root.conf
Queued for next boot: summit.registry/lb1506:bootc-auth
  Version: 9.20240714.0
  Digest: sha256:07eb42017b20ef5f33945014d0be92b077cb4890a97a5def117a745567cbd3f1
Total new layers: 72    Size: 972.0 MB
Removed layers:   3     Size: 127.2 MB
Added layers:     5     Size: 127.2 MB
Rebooting system

Connection to qcow-vm closed by remote host.
```

```
Connection to qcow-vm closed.
```

5.4. Testing the changes

We can check for our new web page from the lab host (`[lab-user@hypervisor rh-summit-2024-lb1506]$`):

```
curl http://qcow-vm
```

Now the output should be "Hello Red Hat Summit Connect 2024!!"

Chapter 6. Repurposing a bootc host

In addition to simplifying updates and providing native rollback, operating RHEL in image mode also makes it simple to quickly change the purpose of a running system. This means experimenting with new versions of application components or testing OS updates is as simple as applying any other image change.

In this lab we'll explore this feature as well as expand on the ideas of layering common in the application container world.

6.1. Writing advanced containerfiles

All of the files for this exercise are provided in the examples folder and we'll be operating on those directly. If a command doesn't operate, or a file doesn't look correct, check to make sure you are in the right directory.

Instead of just running a webserver, what if we needed to run WordPress instances, and our developers need those to be containerized? Image mode hosts are suitable for a wide range of use cases beyond directly hosting applications, and acting as a container host is one of those. In fact, the work that preceeded image mode was largely focused on the role of container hosting.

On the host system (make sure the prompt shows `[lab-user@hypervisor rh-summit-2024-lb1506]$`), you can change to the `wordpress-quadlet` directory to examine the new Containerfile.

```
cd examples/wordpress-quadlet
cat Containerfile.wp
```

Using container tooling to create images let's us take advantage of layers and inheritance in our standard build process. Different teams and work directly from images built by others, rather than by integrating after the fact. Notice our `FROM` line is the image you've been working on throughout the lab. This means that every bit of customization and software you've installed previously will be available on the host built from this new definition.

```
FROM summit.registry/lb1506:bootc

RUN dnf -y install lynx

ADD etc/ /etc
ADD usr/ /usr
```

For an 'advanced` configuration, this doesn't have a lot of content, what's happening?

Remember the `ADD` directive pulls full directories into the image at build time. So all of the work here is

somehow being done in `/usr` or `/etc`, let's see what's in there.

```
ls -lahR etc/ usr/
```

The advanced part is the use of quadlets to run containers. A full description of quadlets is outside the scope of this workshop, but in short, a quadlet is a way to run a container (or group of containers in this case) as a systemd service. In `wordpress-quadlet/etc/` we have a configuration file for the caddy server, a Golang HTTPS server acting as a proxy, and a file of environment variables to be passed to the quadlet.

In `wordpress-quadlet/usr/share/containers/systemd/` we have all of the files that define the quadlet. The `.container` file defines each of the containers for systemd to run. These are typical systemd unit files, aside from the `[Container]` block which is unique to quadlets.

Feel free to explore these files and directories before moving on.

6.2. Build and push the image

When we build this image, we will use a new tag. You may have noticed in the previous exercises, the name of the image has stayed the same. Tags are how `bootc` keeps track of images, which is why each of the previous changes we made showed up as updates.

In this exercise, we're going to look at a different way to change what operating system in installed on a host, so this image will get a new tag. Tagging is a powerful but somewhat subjective way to both provide information about an image and to control what's visible to any particular image mode host. Most of that discussion is out of scope for this lab, but we'll explore how to use an alternate image here. Remember the format for the `--tag` flag is `<registry>/<repository>:<tag>`.

```
podman build --file Containerfile.wp --tag summit.registry/lb1506:bootc-wp
```

And of course push it to the local registry:

```
podman push summit.registry/lb1506:bootc-wp
```

Notice that even though we used a new tag for this image, the push still used cached layers. This is an advantage of stacking images in a standard build design.

You can now login to the virtual machine:

```
ssh lab-user@qcow-vm
```

6.3. Switch and test the image

After the new container image has been pushed to the local registry, you can `switch` the bootc image to the WordPress one. This `bootc` command is how we change what image to follow for updates. From here on, any changes made to the orginal `bootc` tagged image would not show up as an available update, only changes to the new `bootc-wp` image.

```
sudo bootc switch summit.registry/lb1506:bootc-wp
```

As usual, after the command is done you need to reboot the virtual machine for the changes to take effect. Before doing that, please make sure you are logged in to the virtual machine and not the hypervisor (the prompt should look like `[lab-user@qcow-vm ~]#`):

```
sudo systemctl reboot
```

After a short while, you can log back in to the virtual machine:

```
ssh lab-user@qcow-vm
```

6.4. Troubleshooting layered builds

Check on the status of our newly created quadlet by checking the caddy proxy server.

```
sudo systemctl status caddy.service
sudo journalctl -t caddy
```

```
Jul 24 14:40:30 qcow-vm systemd[1]: caddy.service: Failed with result 'exit-code'.
Jul 24 14:40:30 qcow-vm systemd[1]: Failed to start Caddy Quadlet.

Jul 24 14:40:25 qcow-vm caddy[1347]: Error: cannot listen on the TCP port: listen tcp4
:80:
Jul 24 14:40:27 qcow-vm caddy[1780]: Error: cannot listen on the TCP port: listen tcp4
:80:
```

We weren't prompted for our sudo password, but it looks like our new caddy server couldn't bind to port 80 when it tried to start. What's going on? The answer to both lies in the image we built from.

If we check the status of Apache, we can see that it is indeed running and listening on port 80.

```
sudo systemctl status httpd.service
```

If you look at the original Containerfile, you'll recall we set Apache to start at boot:

```
RUN systemctl enable httpd.service
```

Since local changes to /etc are kept by `bootc` when changing images, httpd stayed enabled on this new host as well. Let's disable it and restart caddy.

```
sudo systemctl disable --now httpd.service
sudo systemctl restart caddy.service
sudo systemctl status caddy.service
```

```
Removed "/etc/systemd/system/multi-user.target.wants/httpd.service".

● caddy.service - Caddy Quadlet
     Loaded: loaded (/usr/share/containers/systemd/caddy.container; generated)
     Active: active (running) since Wed 2024-07-24 14:42:21 UTC; 6s ago
```

It looks like caddy started, let's check to see that it's passing requests to the WordPress container in the quadlet. Curl will dump a mess of HTML and we don't have a GUI, but that's why we installed the Lynx browser

```
lynx localhost
```

You should see the WordPress configuration dialog box. You can hit `Q` (Shift + q) to quit lynx.

Feel free to explore the virtual machine before moving on to the next section.

Before proceeding, make sure you have logged out of the virtual machine:

```
logout
```

Easy updates, rollbacks, and image switching are part of the core improvements to the operation of image mode systems. Layering is an important part of the design of standard builds and can have some downstream effects as well. Just like stacking configuration management, thinking through the idea of layered builds can be powerful.

But those aren't the only advantages or design considerations when thinking about how to best use image mode in your workflows. Let's explore a few more advanced topics next.

Chapter 7. Creating an inner loop for system design

In this exercise, you'll look at how to test without needing to deploy the image to a host.

You should on the lab host (the prompt should look like `[lab-user@hypervisor rh-summit-2024-lb1506]$`), if not log out of the VM now.

Being able to make changes, build, and run an application locally is part of what is called the 'inner loop' of application development. Containers had a big impact on making local development and production run time look and feel identical, improving the inner loop for developers. One advantages of image mode, the use of standard container tools brings this 'inner loop' capability to the design and development of standard operating system builds.

While not every nuance of system behavior can be tested this way (eg SELinux polices), this still can greatly reduce the amount of time usually needed to test changes to configurations, software installations, etc.

7.1. Testing a bootc image as a container

You can launch this bootc container like any other application container, `podman` will start `systemd` by default. Using `--detach` will help ensure that `systemd` is PID1 and the bootc image will initialize as intended when running as a host.

```
podman run --rm --name http-test --detach --publish 80:80 summit.registry/lb1506:bootc
```

Test that it is running:

```
podman ps | grep http-test
```

The output of the command above should resemble:

```
06a7bdb1950b  summit.registry/lb1506:latest    15 seconds ago  Up 16 seconds  0.0.0.0:80-
>80/tcp    http-test
```

We exposed a port from the container on the lab host, so we can test the web server.

```
curl http://localhost/
```

The result of the command above should be the latest index.html we created.

```
Hello Red Hat Summit Connect 2024!!
```

Before we attach to an interactive shell in the running container, let's get the kernel command line from the lab host.

```
cat /proc/cmdline
```

```
BOOT_IMAGE=/boot/vmlinuz root=UUID=60d0e5e3-7217-4c34-b8e0-fd3d9abf0654 ro console=tty0
console=ttyS1,115200n8 biosdevname=0 net.naming-scheme=rhel-9.3 net.ifnames=1
modprobe.blacklist=igb modprobe.blacklist=rndis_host
```

Now we can get our shell in the running container.

```
podman exec -it  http-test /bin/bash
```

```
bash-5.1#
```

Let's look at kernel command line as well as PID1 in the `/proc` filesystem and see what runtime info we have.

```
cat /proc/cmdline ①
cat /proc/1/cgroup ②
cat /proc/1/attr/current ③
```

We can see the kernel command line is taken from the lab host since there's no running kernel in a container, systemd is running as the init system from the cgroup hierarchy but a different SELinux context. Remember that systemd is run as PID1 by default when a bootc image is run as a container, but since it wasn't booted as a host, none of the first boot preparation will be executed.

① `BOOT_IMAGE=/boot/vmlinuz root=UUID=60d0e5e3-7217-4c34-b8e0-fd3d9abf0654 ro console=tty0 console=ttyS1,115200n8 biosdevname=0 net.naming-scheme=rhel-9.3 net.ifnames=1 modprobe.blacklist=igb modprobe.blacklist=rndis_host`

② `0::/init.scope`

③ `system_u:system_r:container_init_t:s0:c472,c516`

You can use this container to experiment with changes, understand dependecies that new software might bring, check that services can be started, and more. While issues we introduced in the switch exercise may not be discovered here, there is still a lot that can be done to speed creation and testing of new operating system builds.

Once you're done testing, go ahead and exit the shell, and stop the running container on the host:

```
exit
podman stop http-test
```

Chapter 8. Exploring alternate user creation methods

In this exercise, you'll explore other ways to create users and add credentials to image mode hosts.

8.1. Embedding emergency credentials into an image

In the previous exercises, we've created users and added ssh keys at install time, either via a blueprint file passed to `bootc-image-builder` or via kickstart to Anaconda. There are many ways and many reasons why handling users would happen at any particular stage of the process, during deployment or after install via network authentication.

In addition to adding users during the creation of a host system, via `bootc-image-builder`, `cloud-init`, or some other means, we can also create users during the creation of the bootc image. This can be useful in certain cases where particular emergency users are needed in environments with limited connectivity. These users and credentials would be tied to the image, which means controlling how secrets are used in the build are important to understand before going down this path. You will want to explore more before making this choice in your own environment.

On the host system (make sure the prompt shows `[lab-user@hypervisor rh-summit-2024-lb1506]$`), and are in the `rh-summit-2024-lb1506` directory

```
cd ~/rh-summit-2024-lb1506
```

8.2. Using podman secrets at build time

We will use an addtional feature of `podman` to get the credentials we want to embed passed to the image safely.

You can now edit the `Containerfile` to match the following:

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

COPY certs/004-summit.conf /etc/containers/registries.conf.d/004-summit.conf

COPY templates/30-auth-system.conf /etc/ssh/sshd_config.d/30-auth-system.conf ①
RUN mkdir -p /usr/ssh ②
RUN --mount=type=secret,id=SSHPUBKEY cat /run/secrets/SSHPUBKEY > /usr/ssh/root.keys &&
chmod 0600 /usr/ssh/root.keys ③

ADD etc/ /etc
```

```
RUN dnf install -y httpd

RUN <<EOF
    mv /var/www /usr/share/www
    sed -i 's-/var/www-/usr/share/www-' /etc/httpd/conf/httpd.conf
EOF

RUN echo "Hello Red Hat Summit Connect 2024!!" > /usr/share/www/html/index.html

RUN systemctl enable httpd.service
```

① `COPY templates···` → adds a drop in for sshd to look for public keys in users global `.keys` file, note this is in `/usr` and tied to the image

② `RUN mkdir -p /usr/ssh` → create the new keys directory in `/usr`

③ `` `RUN --mount=type=secret,id=SSHPUBKEY cat /run/secrets/SSHPUBKEY .. ` `` → mounts the secret passed via build to a file only available at build time, then copies the contents

You can now build the new container image with the following command, note the `--secret` argument. This is the connection to the `RUN` directive in the Containerfile. At build time, `podman` will take the file passed and create a temporary secret which we can mount only during the build process. The `id` of the secret is the filename mounted under `/run/secrets`, which we then can extract the contents for our ssh key.

We're also going to use a new tag for this image, like we did in the previous `bootc switch` exercise.

```
podman build --secret id=SSHPUBKEY,src=/home/lab-user/.ssh/id_rsa.pub --file
Containerfile --tag summit.registry/lb1506:bootc-auth
```

And make sure to push it to the registry:

```
podman push summit.registry/lb1506:bootc-auth
```

8.3. Changing the virtual machine image

The virtual machine you have been working with during this workshop should still be running. You can check this with

```
virsh --connect qemu:///system list
```

And the output should contain a virtual machine called `qcow`.

Now you can ssh into the virtual machine

```
ssh lab-user@qcow-vm
```

We can change to our new image with the embedded `root` credentials

```
sudo bootc switch summit.registry/lb1506:bootc-auth
```

The output should look something like this:

```
layers already present: 68; layers needed: 7 (126.7 MB)
 452 B [□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□] (0s) Fetched layer sha256:2691e6642975
Pruned images: 0 (layers: 0, objsize: 58.9 MB)
Queued for next boot: summit.registry/lb1506:bootc-auth
  Version: 9.20240721.0
  Digest: sha256:2afd495b9dd3e72aa4926deb1e738af54e0b1bb935803bc6c107bdd919347167
```

As usual, after the command is done you need to reboot the virtual machine for the changes to take effect. Before doing that, please make sure you are logged in to the virtual machine and not the hypervisor (the prompt should look like `[lab-user@qcow-vm ~]#`):

```
sudo systemctl reboot
```

8.4. Testing the changes

You can now login to the virtual machine with the newly added root credentials:

```
ssh root@qcow-vm
```

And check once again the status of bootc (no need to use `sudo`, you are root):

```
bootc status
```

The output should look like this:

```
apiVersion: org.containers.bootc/v1alpha1
kind: BootcHost
metadata:
  name: host
spec:
  image:
```

```
    image: summit.registry/lb1506:bootc-auth
    transport: registry
  bootOrder: default
status:
  staged: null
  booted:
    image:
      image:
        image: summit.registry/lb1506:bootc-auth
        transport: registry
      version: 9.20240721.0
      timestamp: null
      imageDigest:
sha256:2afd495b9dd3e72aa4926deb1e738af54e0b1bb935803bc6c107bdd919347167
    cachedUpdate: null
    incompatible: false
    pinned: false
    ostree:
      checksum: 6307d9fdc8be3f0c202ce49e311b1d046d6d606dea7bcff51ad5d8910f7887b3
      deploySerial: 0
  rollback:
    image:
      image:
        image: summit.registry/lb1506:bootc-wp
        transport: registry
      version: 9.20240721.0
      timestamp: null
      imageDigest:
sha256:af3c6bd4ae7beb9bb10f9846c7bc9d9ba1ede259b4b45e581ddcd97ab2df4380
    cachedUpdate: null
    incompatible: false
    pinned: false
    ostree:
      checksum: 543f6c9a5b8e2f354db71adda5db7ca74b53bcb9fdf8c8fcd391e7c66c77062a
      deploySerial: 0
  rollbackQueued: false
  type: bootcHost
```

Feel free to explore the virtual machine before moving on to the next section, remembering you are now `root`.

Since we've moved away from the WordPress image back to a standard httpd service, you can look for the original index file on disk or via `curl`.

Before proceeding, make sure you have logged out of the virtual machine:

```
logout
```

Chapter 9. Creating an install iso with Anaconda

In this lab you will learn the basics of deploying a bootc image using Anaconda and an all in one `iso` image.

Anaconda is the official Red Hat Enterprise Linux installer and it uses Kickstart as it's scripting language. Recently, Kickstart received a new command called `ostreecontainer`.

9.1. Building an installation ISO image with the bootc container

Anaconda is typically used via the boot iso shipped by Red Hat. There are several ways to get a host to boot from this ISO image including PXE or HTTP Boot over the network, as an image via IMPI, or a phyiscal disk inserted into a drive. The kickstart file for an install can also be provided in several ways, including as part of the iso or over a network. For this lab, we'll focus on creating a boot ISO that includes the kickstart, mainly for local logistics reasons. The configuration of Anaconda via kickstart will be the same, regardless of how they are presented to the host being installed.

The starting iso image should have been already downloaded at the setup phase at the beginning of this workshop. You can check it exists by listing the file `/var/lib/libvirt/images/summit/rhel-boot.iso` as root:

```
sudo ls -lah /var/lib/libvirt/images/summit/rhel-boot.iso
```

If it is missing, you can redownload it by calling `make iso-download`.

Next, make sure you have pushed the bootc container image to the local registry:

```
podman push summit.registry/lb1506:bootc
```

The actual generation of the custom `iso` image is out of scope for this workshop, so we are providing a make command to do so. This custom ISO will contain not the original boot image, but our kickstart file and the bootc image we want installed on the host. These sort of all in one installers can be very useful for bare metal, especially in places with limited or no network connectivity.

```
make iso CONTAINER=summit.registry/lb1506:bootc
```

At the end of the run, you should be able to list the file `/var/lib/libvirt/images/summit/rhel-boot-custom.iso` as root:

```
sudo ls -lah /var/lib/libvirt/images/summit/rhel-boot-custom.iso
```

For reference, the script used to embed and build the custom ISO can be found in `bin/embed-container` and it can be replicated with the following steps (**no need to run these commands**):

1. use `skopeo` to copy the container image to a temporary directory on disk:

   ```
   skopeo copy "docker://summit.registry/lb1506:bootc" "oci:${TEMPDIR}/container/"
   ```

2. generate a Kickstart file (more details below)
3. create the new iso using `mkksiso`

   ```
   mkksiso --ks local.kickstart --add ${TEMPDIR}/container/ original.iso custom.iso
   ```

9.2. Exploring the kickstart file

The Kickstart added to the iso is in `config/local.ks` and looks like this.

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

ostreecontainer --url=/run/install/repo/container --transport=oci

firewall --disabled
services --enabled=sshd
sshkey --username lab-user "SSHKEY"
user --name="lab-user" --groups=wheel --plaintext --password=lb1506
rootpw lb1506
poweroff
```

The new `ostreecontainer` command replaces the standard %packages block of the kickstart file that defines what RPMs would be installed by Anaconda. This replaces that set of RPM transactions with the `bootc` image to be unpacked and installed to disk.

- `--url=/run/install/repo/container` → path to the local container in the iso but it can also directly take a publicly available registry
- `--transport=oci` → the format in which the container is available, in this case `oci`

If you wanted to deploy directly from a container registry, the `ostreecontainer` command would look like this (**no need to run this command**):

```
ostreecontainer --url=quay.io/myorg/myimage:mytag
```

All of the rest of the kickstart is standard fare. Anaconda will create the partitions and filesystems on disk, enable services, and create users just like it would in any other install. You even have access to the `%pre` and `%post` blocks too, the only missing section from any kickstart you have today is the `%packages` list.

Once you see the following output, you can proceed.

```
ISO image produced: 1028128 sectors
Written to medium : 1028288 sectors at LBA 32
Writing to '/var/lib/libvirt/images/summit/rhel-boot-custom.iso' completed successfully.

Done
```

9.3. Starting the installation with Anaconda in a virtual machine

Having created a custom installation ISO, we can boot a virtual machine and automatically install our image. Creating a virtual machine from scratch is out of scope for this workshop, so we have provided a make command. We are once again using `virt-install` to create a local VM, the main difference is that here we are using the `--location` option to pass the local path of the ISO file to the install process. This emulates inserting a phyiscal disk into a bare metal system.

```
make vm-iso
```

The command above will start the installation process and you should see the console of a newly created virtual machine booting from the custom iso. Watch for the Anaconda output as it proceeds with the unattended install.

```
Powering off.
[   65.973246] reboot: Power down

Domain creation completed.
You can restart your domain by running:
  virsh --connect qemu:///system start iso
virsh --connect "qemu:///system" start "iso"
Domain 'iso' started
```

You can now see if the virtual machine is running:

```
virsh --connect qemu:///system list
```

```
 Id   Name   State
----------------------
 1    qcow   running
 4    iso    running
```

9.4. Test and login to the virtual machine

Like with the previous virtual machine created, you can directly see if the http application is already running on the host:

```
curl http://iso-vm
```

The output should be "Hello Red Hat Summit Connect 2024!!"

You should also be able to login to the virtual machine:

```
ssh lab-user@iso-vm
```

If the ssh key is not automatically picked up, use the password `lb1506`.

You can now check the status of `bootc`:

```
sudo bootc status
```

The output should be similar to this:

```
apiVersion: org.containers.bootc/v1alpha1
kind: BootcHost
metadata:
  name: host
spec:
  image:
    image: /run/install/repo/container
    transport: oci
  bootOrder: default
status:
```

```
  staged: null
  booted:
    image:
      image:
        image: /run/install/repo/container
        transport: oci
      version: 9.20240501.0
      timestamp: null
      imageDigest:
sha256:0a3daed6e31c2f2917e17ea994059e1aaee0481fe16836c118c5e1d10a87365c
    cachedUpdate: null
    incompatible: false
    pinned: false
    ostree:
      checksum: 42f36e87a9436d505b3993822b92dbf7961ad3f1a8fddf67b91746df365784f0
      deploySerial: 0
  rollback: null
  rollbackQueued: false
  type: bootcHost
```

9.5. Switching to a different transport method

One thing that immediately is different in the `bootc status` output is that the deployed image image is a local path, not the container naming convention we've been using:

```
spec:
  image:
    image: /run/install/repo/container
    transport: oci
  bootOrder: default
```

The `transport` line refers to the OCI definition of images which includes how they are pulled. The `oci` transport means this is a single image located at a specific local path. This is useful for installing the way we did, but less so for updates.

So far in this lab, we have been using the `registry` transport, which requires network access. If we wanted to manage updates in an offline manner, say for disconnected environments or those with intermittent connectivity, we can use `containers-storage` which refers to the locally configured shared locations. A full discussion of transports and their associated uses and configuration is outside the scope of this lab.

To keep with the theme of offline usage, let's simulate updating from the local storage. We can use `skopeo` to copy images from one location to another. In the embedded ISO script, it's used to copy from the registry to be included in the final image. Here, we can use it to copy from the registry to the host. We need to be sure to use `sudo` to copy into the system storage location and not the user's.

```
sudo skopeo copy --tls-verify=false docker://summit.registry/lb1506:bootc  containers-
storage:summit.registry/lb1506:bootc
```

Switch our installation to use the new container image, using the `--transport` flag to let bootc know we want to use local container storage for this operation.

```
sudo bootc switch --transport containers-storage summit.registry/lb1506:bootc
```

```
Loading usr/lib/ostree/prepare-root.conf
Queued for next boot: ostree-unverified-image:containers-
storage:summit.registry/lb1506:bootc
  Version: 9.20240501.0
  Digest: sha256:0a3daed6e31c2f2917e17ea994059e1aaee0481fe16836c118c5e1d10a87365c
```

At this point, the "new" installation has been prepared and will be started at next boot of the virtual machine.

The last step for the change to take is to reboot the virtual machine. Before doing it, please make sure you are logged in to the virtual machine and not the hypervisor (the prompt should look like `[lab-user@lb1506-vm ~]$`):

```
sudo systemctl reboot
```

In a short time after that command, you should be able to ssh back to the virtual machine:

```
ssh lab-user@iso-vm
```

And check the bootc status:

```
sudo bootc status
```

```
apiVersion: org.containers.bootc/v1alpha1
kind: BootcHost
metadata:
  name: host
spec:
  image:
    image: summit.registry/lb1506:bootc
    transport: containers-storage
```

```
  bootOrder: default
status:
  staged: null
  booted:
    image:
      image:
        image: summit.registry/lb1506:bootc
        transport: containers-storage
      version: 9.20240501.0
      timestamp: null
      imageDigest:
sha256:0a3daed6e31c2f2917e17ea994059e1aaee0481fe16836c118c5e1d10a87365c
    cachedUpdate: null
    incompatible: false
    pinned: false
    ostree:
      checksum: 6e468a048b5c86ed8c481040b125b442b9222c914fc12799123717eb94fc43b6
      deploySerial: 0
  rollback:
    image:
      image:
        image: /run/install/repo/container
        transport: oci
      version: 9.20240501.0
      timestamp: null
      imageDigest:
sha256:0a3daed6e31c2f2917e17ea994059e1aaee0481fe16836c118c5e1d10a87365c
    cachedUpdate: null
    incompatible: false
    pinned: false
    ostree:
      checksum: 42f36e87a9436d505b3993822b92dbf7961ad3f1a8fddf67b91746df365784f0
      deploySerial: 0
  rollbackQueued: false
  type: bootcHost
```

In the status you can see `bootc` is now tracking local container storage for updates. Further updates would then be provided on media presented to the host, like a USB drive or DVD. You could use skopeo sync a registry repository to media as well as copy it from the media to the local storage on the host. These images are visible to podman as well.

```
sudo podman images
```

There are a range of possibilities for edge devices, disconnected networks, and any other arenas where direct connectivity to a registry over a network isn't possible or desired.

9.6. Workshop complete!

You've now completed all of the exercises in today's workshop.

Image mode for RHEL provides a new way to think about risks involved with updating hosts, creates native rollback functionality, and can quickly and easily change the role of a particular host. We hope you've had a few ideas of how the techiniques and topics in this workshop could apply to the environments you manage.