Embedded-Matt's Upstream Image Mode Lab Exercises

V1.0   October 4, 2024

embedded-matt@redhat.com

Technical Requirements

You will need two bare metal or virtual machines that you can modify or reinstall the operating system upon. Root access, Internet access, and DHCP IP addressing are a must-have. You will also need the ability to download ISO images and have a 16GB (or greater) USB thumb drive. Finally, you will need a free Red Hat Developer account and access to your own Quay registry (also free). The requirements are greater for these exercises as the outcomes are more significant. I am hopeful you'll even have fun along the way, which in my opinion is also a requirement.

For these exercises. We will use CentOS Streams 9 as our build machine's OS, and we'll be creating a CentOS Streams 9 bootable container image.

Exercise 1: Prepare Environment

In this first exercise, we will install necessary tools along with some optional tools to create our minimal bootable container image build chain.

1. First, we will setup a build environment, configure our registry, and create a container that will become the basis for not just our application but our operating system as well.

(command)

```
$ sudo dnf install -y containers-common crun iptables netavark \
nftables slirp4netns composer-cli cockpit cockpit-composer \
skopeo buildah runc podman
```

(output suppressed to save space)

2. Now we'll ensure that the web console is enabled.

(command)

```
$ sudo systemctl enable cockpit.socket
```

(no output)

3. And here, we'll start the socket for the web console.

(command)

```
$ sudo systemctl start cockpit.socket
```

(no output)

4. Confirm the web console is active.

(command)

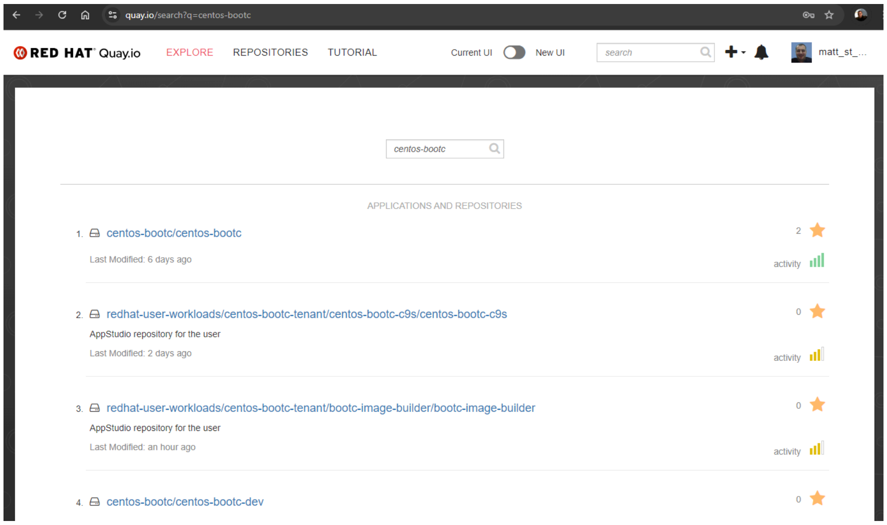
```
$ sudo systemctl status cockpit.socket
```

(output)

```
● cockpit.socket - Cockpit Web Service Socket
     Loaded: loaded (/usr/lib/systemd/system/cockpit.socket; enabled; preset: disabled)
     Active: active (listening) since Fri 2024-08-09 12:39:24 EDT; 1 month 13 days ago
   Triggers: ● cockpit.service
       Docs: man:cockpit-ws(8)
     Listen: [::]:9090 (Stream)
      Tasks: 0 (limit: 38320)
     Memory: 648.0K (peak: 2.5M)
        CPU: 26ms
     CGroup: /system.slice/cockpit.socket

Aug 09 12:39:24 bm03.local systemd[1]: Starting cockpit.socket - Cockpit Web Service Socket...
Aug 09 12:39:24 bm03.local systemd[1]: Listening on cockpit.socket - Cockpit Web Service Socket.
```

(end output)

5. Next we'll take a look at our free registry and make some configuration changes.
    a. In a browser window log into https://quay.io/
    b. Navigate to Account>>Settings>>CLI Password
    c. Set a CLI password if you have not already done so. (you may be asked to create an encrypted password - these are better) Make note of this information we will need it very soon.
    d. Search for repository "centos-bootc"
    e. Click on the link for **centos-bootc/centos-bootc**.
    f. Make note of the URL, as we'll be using it soon. Consider bookmarking the page too.



APPLICATIONS AND REPOSITORIES

1. centos-bootc/centos-bootc — 2 ★
   Last Modified: 6 days ago — activity
2. redhat-user-workloads/centos-bootc-tenant/centos-bootc-c9s/centos-bootc-c9s — 0 ★
   AppStudio repository for the user
   Last Modified: 2 days ago — activity
3. redhat-user-workloads/centos-bootc-tenant/bootc-image-builder/bootc-image-builder — 0 ★
   AppStudio repository for the user
   Last Modified: an hour ago — activity
4. centos-bootc/centos-bootc-dev — 0 ★

Searching for CentOS Stream’s boot_c base image

6. Back in your terminal, configure your non-root user account to be able to search the registry.

(command)
$ cd ~
(no output)

7. Create the requisite directories to store your container configuration so podman will know where to seek information.

(command)
$ mkdir -p ~/.config/containers
(no output)

8. Let’s change directories to the one we just created.

(command)
$ cd ~/.config/containers
(no output)

9. Let’s now create a registries.conf file to contain the defined contents below.

(command)
$ vi registries.conf

Set the file’s contents to match the following and save the file

```
# we will use these registries only

[registries.search]
registries = [‘registry.redhat.io’,’quay.io’]
```

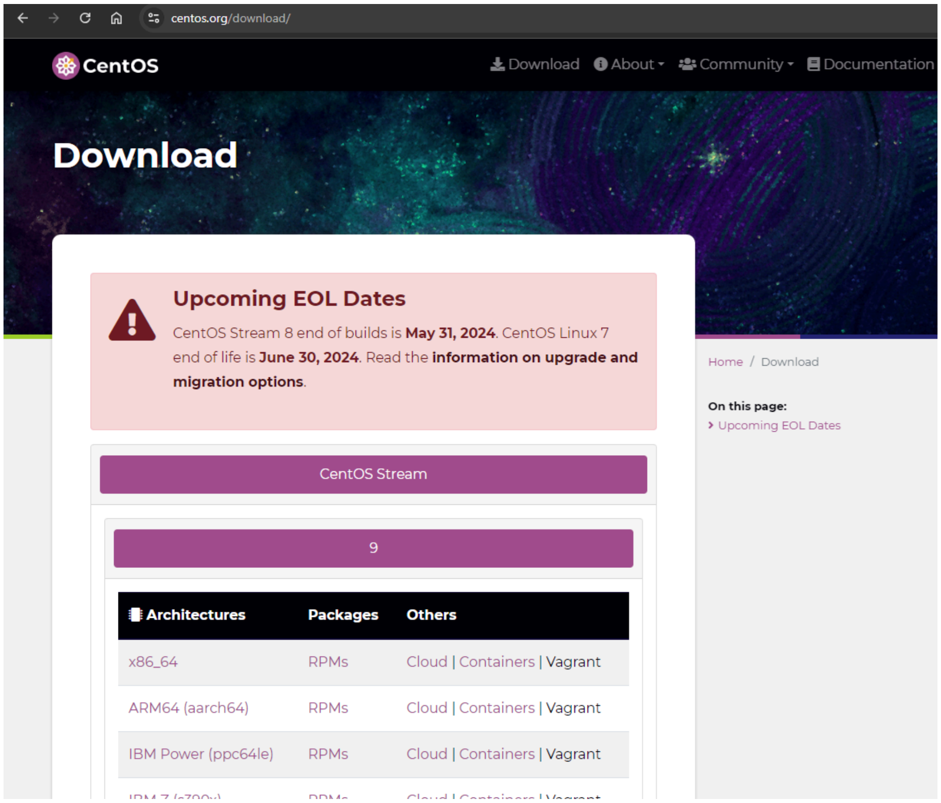
10. Now, we'll go back to our home directory.

(command)

```
$ cd ~
```

(no output)

11. Let's download Centos Streams 9 DVD ISO image to be used later in the process when we create our custom installer. Use your web browser to go to https://centos.org/download/ and then click on the **x86_64** button.



Upcoming EOL Dates

CentOS Stream 8 end of builds is **May 31, 2024**. CentOS Linux 7 end of life is **June 30, 2024**. Read the **information on upgrade and migration options**.

Architectures	Packages	Others
x86_64	RPMs	Cloud | Containers | Vagrant
ARM64 (aarch64)	RPMs	Cloud | Containers | Vagrant
IBM Power (ppc64le)	RPMs	Cloud | Containers | Vagrant
IBM Z (s390x)	RPMs	Cloud | Containers | Vagrant

CentOS Streams 9 Download

12. Let's verify the file size and that it's fully downloaded.

(command)

```
$ ls -lh | grep *.iso
```

(output)

```
-rw-r--r--. 1 mstonge mstonge  11G Sep 25 01:01 CentOS-Stream-9-latest-x86_64-dvd1.iso
```

(end output)

13. Back in your terminal, login to quay.io via command line. Use your own user account and password you have previously set in order to access the Quay.io registry.

(command)

```
$ podman login quay.io
```

(output)

```
Login Succeeded!
```

(end output)

NOTE: If you opted to generate the encrypted CLI password the login command should look a little different. Replace your own username where ***[your username]*** is called out and your long encrypted password where the ***[encrypted pass]*** is called out below.

```
$ podman login -u=’[your username]’ -p=’[encrypted pass]’ quay.io
```

14. Let’s pull down our base container image

(command)

```
$ podman pull quay.io/centos-bootc/centos-bootc:stream9
```

(output)

```
((( output truncated)))

Copying blob 775d29f76a39 done   |
Copying blob 7eff373befa3 done   |
Copying blob 8c789e616763 done   |
Copying blob fd730fb4a24b done   |
Copying blob 54246c915569 done   |
Copying blob 232fb94490b0 done   |
Copying blob ad312c5c40cc done   |
Copying blob bd9ddc54bea9 done   |
Copying config a1163a9d15 done   |
Writing manifest to image destination
a1163a9d15d2f9a3f7f81748baf8fbcfc69690ed38030e770fe2006c090b0f83
```

(end output)

15. Let's check our local container inventory and verify we have the intended CentOS Stream 9 boot_c image amongst our inventory.

(command)

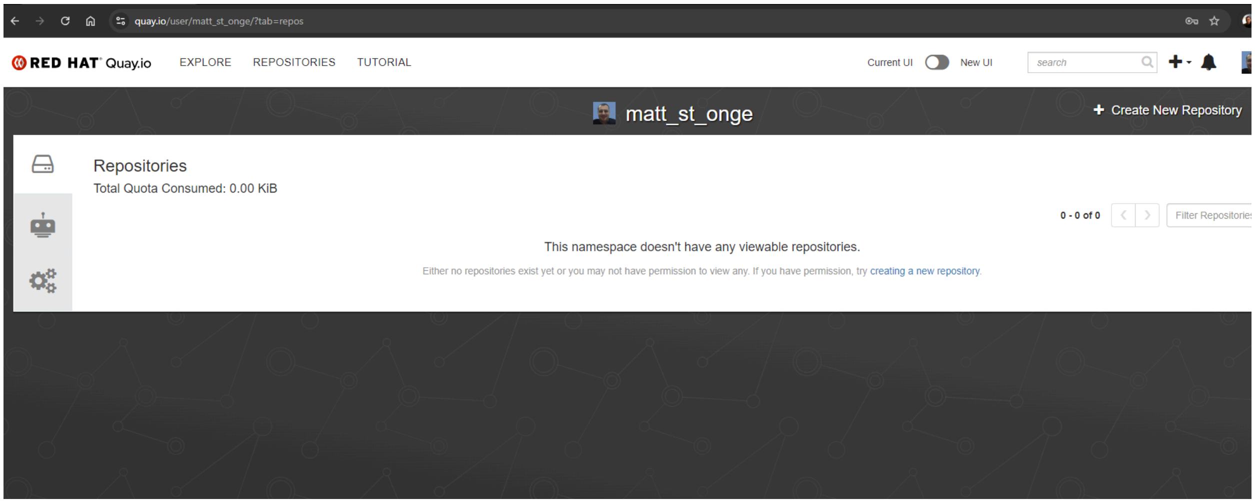
$ podman images

(output)

```
REPOSITORY                          TAG        IMAGE ID      CREATED        SIZE
quay.io/centos-bootc/centos-bootc  stream9     a1163a9d15d2  43 hours ago  1.52 GB
```

(end output)

16. Now in the Quay.io web interface we will create our own public repository for our container image project. We'll start on the Repositories tab.
NOTE: As a simple reminder, this is not exactly how one should ever do this in production. We'll be using a public configuration for lab purposes only. Private, secure registries are the only way I would recommend doing this deployment method whilst being connected to the internet.



quay.io/user/matt_st_onge/?tab=repos

Repositories
Total Quota Consumed: 0.00 KiB

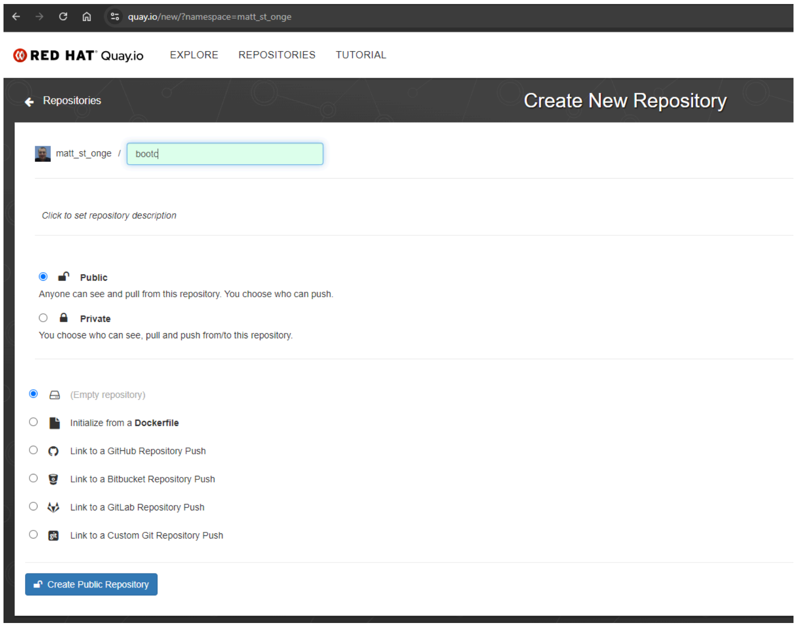
This namespace doesn't have any viewable repositories.

Either no repositories exist yet or you may not have permission to view any. If you have permission, try creating a new repository.

Quay - Repositories main page

17. Next we'll click on the "**+ Create New Repository**" button in the top right hand side of the screen.

Name your repository "bootc" , ensure that the **Public** radio button is selected along with the "**empty repository**" radio button also being selected. Then click the "**Create Repository**" button at the bottom.



quay.io/new/?namespace=matt_st_onge

matt_st_onge / bootc

Click to set repository description

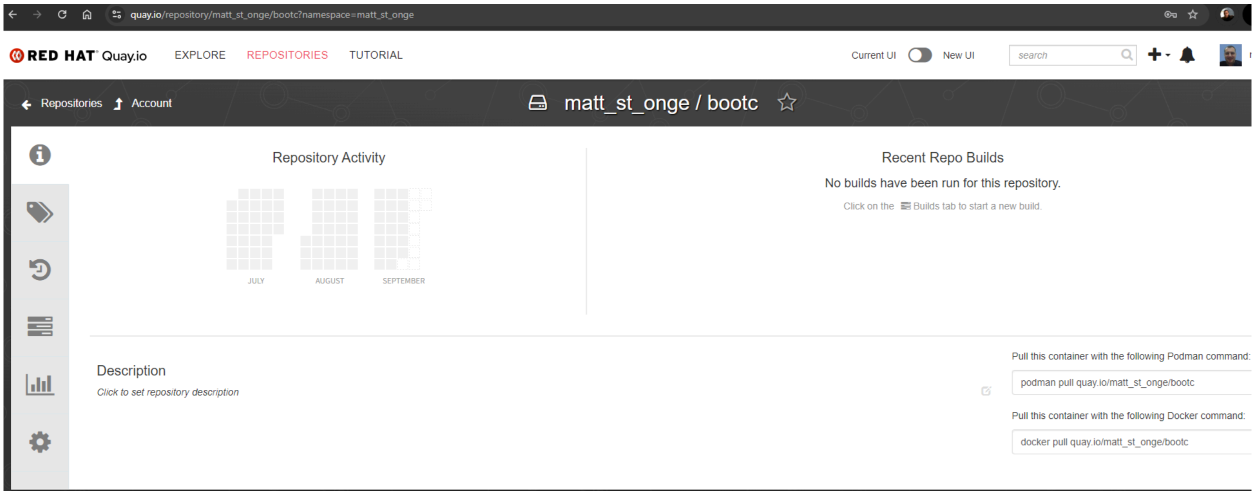
● Public
Anyone can see and pull from this repository. You choose who can push.

○ Private
You choose who can see, pull and push from/to this repository.

● (Empty repository)
○ Initialize from a **Dockerfile**
○ Link to a GitHub Repository Push
○ Link to a Bitbucket Repository Push
○ Link to a GitLab Repository Push
○ Link to a Custom Git Repository Push

Creating a new repository in Quay

Make note of the URL for your repository, we'll be using it soon. I also recommend bookmarking it in your web browser too.



quay.io/repository/matt_st_onge/bootc?namespace=matt_st_onge

matt_st_onge / bootc

Repository Activity

Recent Repo Builds
No builds have been run for this repository.
Click on the Builds tab to start a new build.

Description
Click to set repository description

Pull this container with the following Podman command:
podman pull quay.io/matt_st_onge/bootc

Pull this container with the following Docker command:
docker pull quay.io/matt_st_onge/bootc

Your custom Quay repository

Now we have configured a baseline build environment on your system. By configuring how we leverage registries, installed container tools, and staged a container base image and a Linux installer ISO image, we have all we need to be successful. We additionally setup the web console on our build system that can come in handy later on. Let's move onto building our initial container which will be the basis for our future immutable image.

Exercise 2: Create a Container File

In this exercise, we'll create a container file that we will build using the base image we downloaded in the previous exercise. The results of this exercise will give us a container that can run a simple LAMP (Linux Apache mySQL and PHP) stack. If you decide not to type this file, there will be a prebuilt one in the book's GitHub for reference.

1. Let's create the containerfile and name it **mycontainerfile.cf** .

   (command)

   ```
   $ vi mycontainerfile.cf
   ```

   (end command)

   Set the contents to look like the following:

   ```
   FROM quay.io/centos-bootc/centos-bootc:stream9

   #install the lamp components

   RUN dnf install -y httpd mariadb mariadb-server php-fpm php-mysqlnd && dnf clean all

   #start the services automatically on boot

   RUN systemctl enable httpd mariadb php-fpm

   #create an awe inspiring home page (all one command line)

   RUN echo '<h1 style="text-align:center;">Welcome to My Appliance</h1> <?php phpinfo(); ?>' >>
   /var/www/html/index.php
   ```

2. Let's build our container image.

   NOTE: Replace your own Quay.io username where *[my_account]* appears in the command line.

   (command)

   ```
   $ podman build -f mycontainerfile.cf \
   -t  quay.io/[my_account]/bootc/lamp-bootc:latest
   ```

   (begin output)

   ```
   STEP 1/4: FROM quay.io/centos-bootc/centos-bootc:stream9
   ```

```
STEP 2/4: RUN dnf install -y httpd mariadb mariadb-server php-fpm php-mysqlnd && dnf
clean all
--> Using cache
a525b1bb126820c8522199f6d42b292210f06e4d178efbc148d97a92b94a64ed
--> a525b1bb1268
STEP 3/4: RUN systemctl enable httpd mariadb php-fpm
--> Using cache
9400a8bbc0287454ae0db9f42f9b49e518daf2b410fd3c3d0bb91a8b58e0a2a3
--> 9400a8bbc028
STEP 4/4: RUN echo '<h1 style="text-align:center;">Welcome to My Appliance</h1>
<?php phpinfo(); ?>' >> /var/www/html/index.php
--> Using cache
4bcb220e3de6429f9f83264e84f064f2101c715c78e1104e388d11f6007b560e
COMMIT quay.io/matt_st_onge/bootc/lamp-bootc:latest
--> 4bcb220e3de6
Successfully tagged quay.io/matt_st_onge/bootc/lamp-bootc:latest
4bcb220e3de6429f9f83264e84f064f2101c715c78e1104e388d11f6007b560e
```

(end output)

3. Great! We now have our container image. Let's do a quick test to see how well it works. **NOTE: Replace your own Quay.io username where *[my_account]* appears in the command line.**

   (command)

   ```
   $ podman run -d —rm —name lamp -p 8080:80 \
   quay.io/[my_account]/bootc/lamp-bootc:latest
   ```

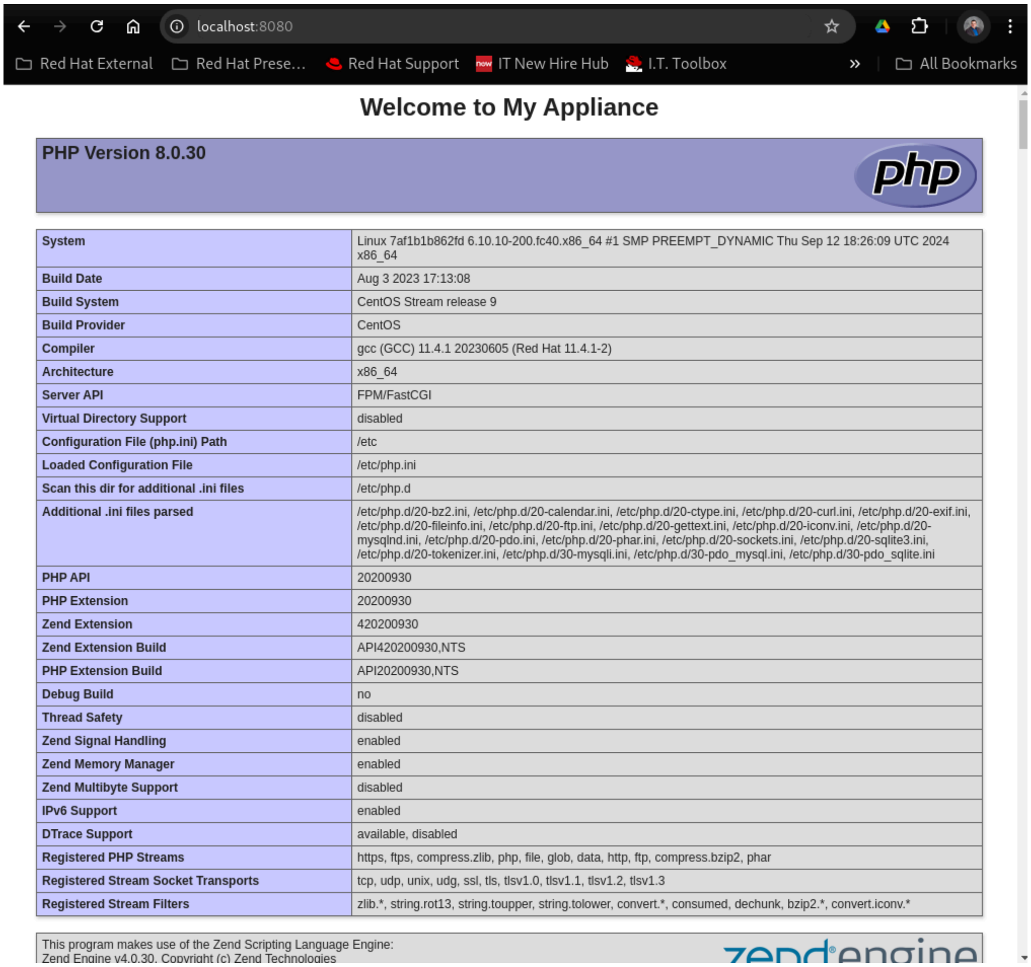
   Your output will resemble something like this following line.

   (begin output)

   ```
   7d9c474d9dd4e6ab32d910c72775cdb111adfed764f29887c110461ca67c54a6
   ```

   (end output)

4. With the container started, let's open a browser window and verify that you can view the served content. http://[your_ip_address]:8080 If the page doesn't load, double check your firewall settings. If you are on the same system where you are running the container, your loopback address should also work.



Welcome to My Appliance

PHP Version 8.0.30

System	Linux 7af1b1b862fd 6.10.10-200.fc40.x86_64 #1 SMP PREEMPT_DYNAMIC Thu Sep 12 18:26:09 UTC 2024 x86_64
Build Date	Aug 3 2023 17:13:08
Build System	CentOS Stream release 9
Build Provider	CentOS
Compiler	gcc (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)
Architecture	x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-mysqlnd.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/30-mysqli.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, compress.bzip2, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, convert.*, consumed, dechunk, bzip2.*, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v4.0.30, Copyright (c) Zend Technologies

Testing your container

5. We should now also be able to shell into the container while it's running.
   (command)
   $ podman exec -it lamp /bin/bash

6. When given the prompt feel free to test some commands but remember to EXIT.
   (command)
   bash-5.1# exit

   You will then be returned to your regular shell prompt on your system.

7. Stop the running container since we know that the image works.
   (command)
   $ podman stop lamp
   (begin output)

```
lamp
```

(end output)

8. Our final step will be the saving of our functional container image to your own repository within Quay.io
   NOTE: Log back into quay.io via command line if you are not logged in already.
   NOTE: Replace your own Quay.io username where *[my_account]* appears in the command line.
   (command)
   ```
   $ podman push quay.io/[my_account]/bootc/lamp-bootc:latest
   ```
   (begin output)
   ```
               (((output truncated)))

   Copying blob 7685af3680f8 skipped: already exists
   Copying blob 9046686a9227 skipped: already exists
   Copying blob d1c1676ee4e9 skipped: already exists
   Copying blob 7a1c4a9ce068 skipped: already exists
   Copying blob 0811ec9b544a done   |
   Copying blob abef090ec865 done   |
   Copying blob 6394663daed5 done   |
   Copying blob 2daf40f13a19 skipped: already exists
   Copying blob 9dad063a624b skipped: already exists
   Copying config 8a4585ebc8 done   |
   Writing manifest to image destination
   ```
   (end output)

Excellent! You've created the basis of your future operating system via the base image and layered on the applications stack all by creating a working container image. In our next exercise we'll create an installer so we can deploy it as a bootable image.

Exercise 3: Create an Installer

In this exercise we will create a kickstart file and then take that kickstart along with a standard vendor provided ISO install image and create our own custom ISO installer image for our amazing new system. This method of installation is great when you're working in your lab or datacenter. Alternative methods will be necessary if you are deploying in a cloud services provider. For a great reference on how to build kickstart files, you can check out this guide:

https://docs.fedoraproject.org/en-US/fedora/f36/install-guide/appendixes/Kickstart_Syntax_Reference/ …

1. In this first step you will create a kickstart file (**mykickstart.ks**). Within this file you will substitute your own account username where I state **[you]** and you’ll also be setting basic configuration for the operating system’s filesystem layout and root password. Save and exit the file when you are done. Should you choose not to type the file in its entirety, there’s an example in the book’s GitHub repository.
   NOTE: Replace your own Quay.io username where *[my_account]* appears in the file’s contents.
   (command)
   $ vi mykickstart.ks

Ensure the contents of **mykickstart.ks** look similar (with your substitutions) to this… I’ve added myself as a user, you should replace that line with your own user account information.

```
# mykickstart.ks
# version 1

# anaconda installer type
text

# ensure that you connect your device to a Ethernet network with active DHCP
network -—bootproto=dhcp -—device=link -—activate

# basic partitioning
clearpart —-all -—initlabel -—disklabel=gpt

reqpart --add-boot

part / --grow --fstype xfs

# here’s where we reference the container image
# notice this kickstart has no packages section
ostreecontainer —-url quay.io/matt_st_onge/bootc/lamp-bootc:latest \
 —-no-signature-verification

# additional settings for demonstration purposes
# in production use better settings
# the purpose of this exercise is not to tech you kickstart
# but to show how to leverage it in custom installers

firewall —-disabled
services —-enabled=sshd

# add your own user account to the system
user —-name=mstonge -—groups=wheel —-plaintext —-password=embedded

# set root password
rootpw —-plaintext —-password=embedded
```

2. Now we will install the software which will enable us to create a custom installer ISO image.
   (command)
   ```
   $ sudo dnf install -y lorax
   ```
   (begin output)
   ```

           ((( output truncated)))

   Complete!

   ```
   (end output)

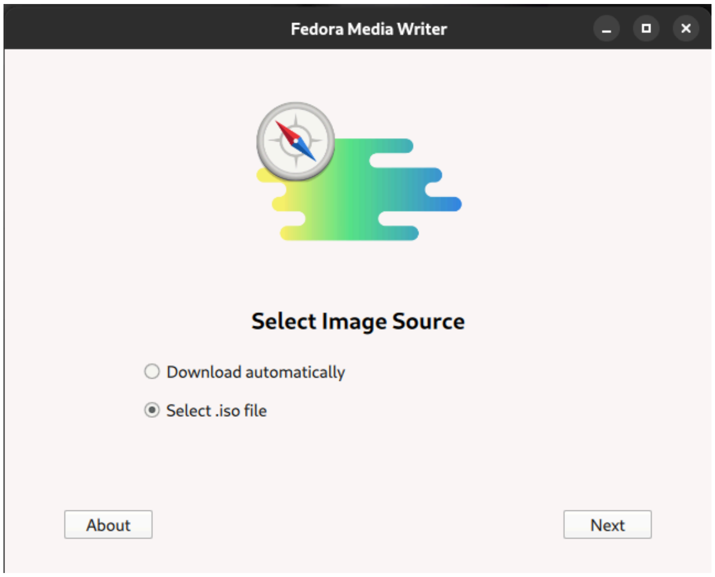
3. Now we'll utilize the **mkksiso** command which is part of the lorax RPM package we just installed. This will create a custom installer for us.
   NOTE: you will need to substitute exact paths of your own here.
   Example: sudo mkksiso --ks /home/mstonge/mykickstart.ks \ /home/mstonge/CentOS-Stream-9-latest-x86_64-dvd1.iso \ /home/mstonge/mycustominstaller.iso
   (command)
   ```
   $ sudo mkksiso –ks [absolute path to mykickstart.ks] \
   [absolute path to the CentOS Stream 9 ISO] \
   [absolute path to the new ISO you want created]
   ```
   (begin output)
   ```

   xorriso 1.5.6 : RockRidge filesystem manipulator, libburnia project.

           ((( output truncated)))

   xorriso : UPDATE : Writing:    344064s  30.7%  fifo  13%  buf  50%  127.5xD
   xorriso : UPDATE : Writing:    442368s  39.5%  fifo  19%  buf  50%  145.1xD
   xorriso : UPDATE : Writing:    548864s  49.0%  fifo  22%  buf  50%  157.3xD
   xorriso : UPDATE : Writing:    644416s  57.5%  fifo   0%  buf  50%  141.1xD
   xorriso : UPDATE : Writing:    737280s  65.8%  fifo  10%  buf  50%  137.1xD
   xorriso : UPDATE : Writing:    830548s  74.1%  fifo   0%  buf  50%  137.6xD
   xorriso : UPDATE : Writing:    932628s  83.2%  fifo   0%  buf  50%  150.7xD
   xorriso : UPDATE : Writing:   1007616s  89.9%  fifo  29%  buf  50%  110.7xD
   xorriso : UPDATE : Writing:   1097728s  97.9%  fifo  10%  buf  50%  133.0xD
   ISO image produced: 1120832 sectors
   ```

```
Written to medium : 1121008 sectors at LBA 48
Writing to '/home/mstonge/mycustominstaller.iso' completed successfully.
```

(end output)

4. Now that we have our own custom installer ISO image, so let’s create boot media. For this step, you will need to use Fedora Media Writer. It should already be on your system, if not, download it first.
   NOTE: this step may be optional if you are working with virtual machines - you might be able just to boot from the ISO file itself within the hypervisor.

Let’s look at how the Fedora Media Writer can simplify the creation of boot media.



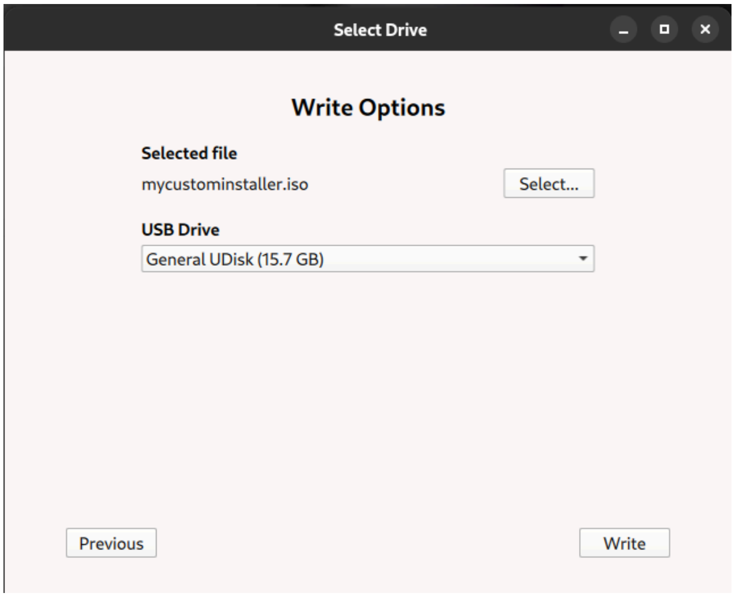
Select Image Source

○ Download automatically

● Select .iso file

Fedora Media Writer

Here, you select the ISO image and the USB thumb drive that you want to commit the bootable image to.



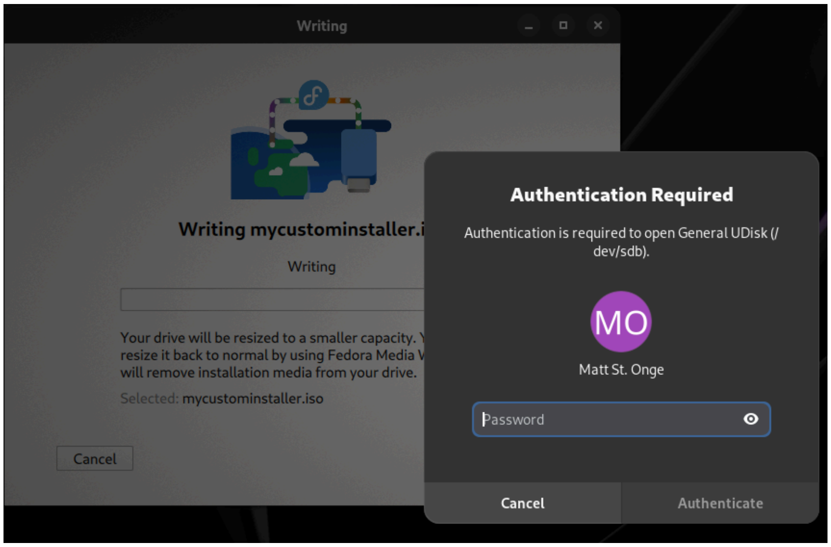
Write Options

Selected file
mycustominstaller.iso

USB Drive
General UDisk (15.7 GB)

Choosing ISO Images

As the Fedora Media Writer requires elevated access, you’ll be prompted for authentication to achieve “sudo” status.



Writing mycustominstaller.i

Writing

Your drive will be resized to a smaller capacity. Y
resize it back to normal by using Fedora Media V
will remove installation media from your drive.

Selected: mycustominstaller.iso

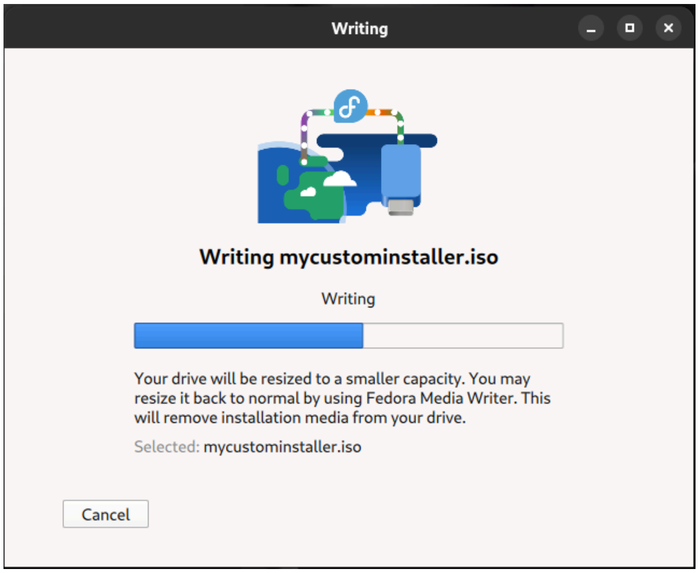
Authentication Required

Authentication is required to open General UDisk (/dev/sdb).

Matt St. Onge

Elevated permissions - authentication

It will definitely take a few minutes to render the ISO image to the physical media. Have patience, grab a beverage, and enjoy the break.



Writing mycustominstaller.iso

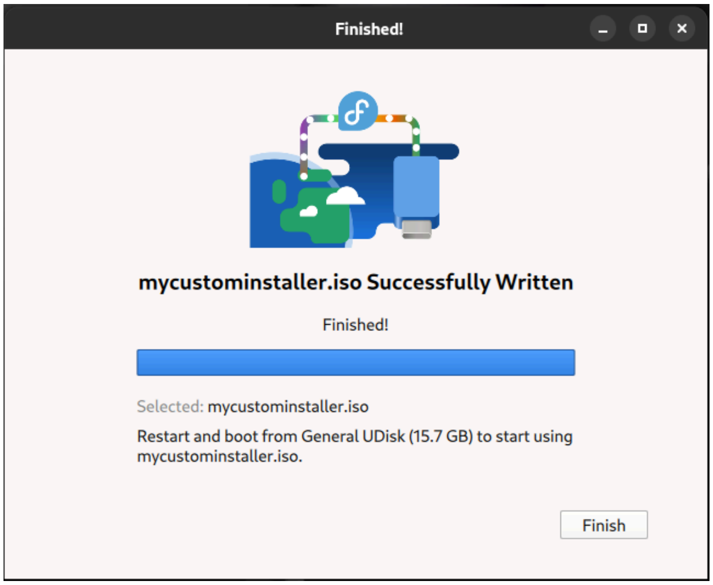
Writing

Your drive will be resized to a smaller capacity. You may resize it back to normal by using Fedora Media Writer. This will remove installation media from your drive.

Selected: mycustominstaller.iso

ISO build in progress

Once completed, you'll be greeted by this screen.



mycustominstaller.iso Successfully Written

Finished!

Selected: mycustominstaller.iso

Restart and boot from General UDisk (15.7 GB) to start using mycustominstaller.iso.

ISO Image create completed

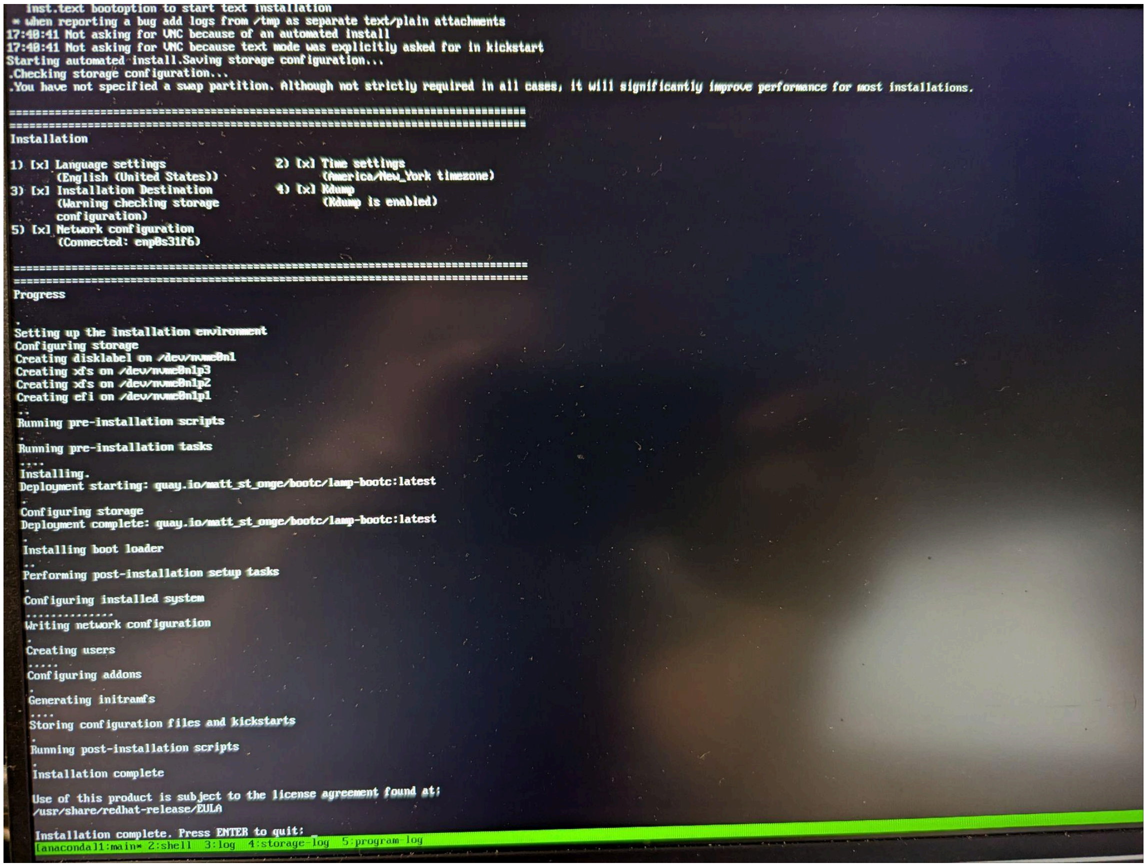
You can now remove the thumb drive from the USB port. We're just moments away from installation. You've successfully created your own custom installer. Let's move on and put it to good use.

Exercise 4: Initial Installation

In this exercise we will install our first Image Mode system with our newly created custom installer. You will boot the test system from the newly created thumbdrive (or from the ISO file we just created in the case of a virtual machine). You may need to interrupt your system's normal boot process to get it to boot from the USB thumb drive.
Sit back, relax, and watch the magic happen…

1. With your newly created boot media, use it to boot (or create a new VM). If booting onto physical hardware, some things to be aware of before your installation:
    a. Ensure all previous partitions are removed from the drive (especially the UEFI partition) before the installation process
    b. Ensure that within your UEFI BIOS any previous entries for Secure Boot are removed (RESET) and that Secure Boot is set to DISABLED before the installation
    c. Boot your system from the USB media (physical hardware) or directly from the ISO image (VM). This is an automated install and it will notify you upon completion (or failure)...

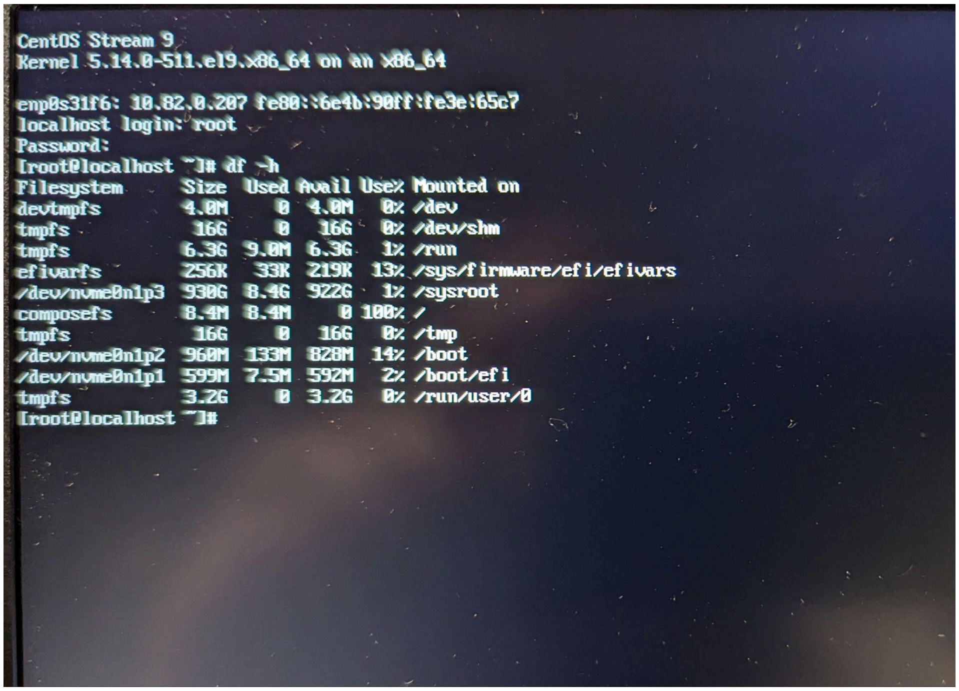
Here's what a successful text-based unattended installation looks like.



```
  inst.text bootoption to start text installation
 * when reporting a bug add logs from /tmp as separate text/plain attachments
17:40:41 Not asking for VNC because of an automated install
17:40:41 Not asking for VNC because text mode was explicitly asked for in kickstart
Starting automated install.Saving storage configuration...
.Checking storage configuration...
.You have not specified a swap partition. Although not strictly required in all cases, it will significantly improve performance for most installations.

================================================================================
================================================================================
Installation

1) [x] Language settings                 2) [x] Time settings
       (English (United States))                (America/New_York timezone)
3) [x] Installation Destination          4) [x] Kdump
       (Warning checking storage                (Kdump is enabled)
       configuration)
5) [x] Network configuration
       (Connected: enp0s31f6)

================================================================================
================================================================================
Progress

.
Setting up the installation environment
Configuring storage
Creating disklabel on /dev/nvme0n1
Creating xfs on /dev/nvme0n1p3
Creating xfs on /dev/nvme0n1p2
Creating efi on /dev/nvme0n1p1
..
Running pre-installation scripts
.
Running pre-installation tasks
....
Installing.
Deployment starting: quay.io/matt_st_onge/bootc/lamp-bootc:latest
.
Configuring storage
Deployment complete: quay.io/matt_st_onge/bootc/lamp-bootc:latest
.
Installing boot loader
..
Performing post-installation setup tasks
.
Configuring installed system
...............
Writing network configuration
.
Creating users
.....
Configuring addons
.
Generating initramfs
....
Storing configuration files and kickstarts
.
Running post-installation scripts
.
Installation complete

Use of this product is subject to the license agreement found at:
/usr/share/redhat-release/EULA

Installation complete. Press ENTER to quit:
[anaconda]1:main* 2:shell  3:log  4:storage-log  5:program-log
```

Installation Success!

2. Once the installation has completed, test your login credentials at the console.



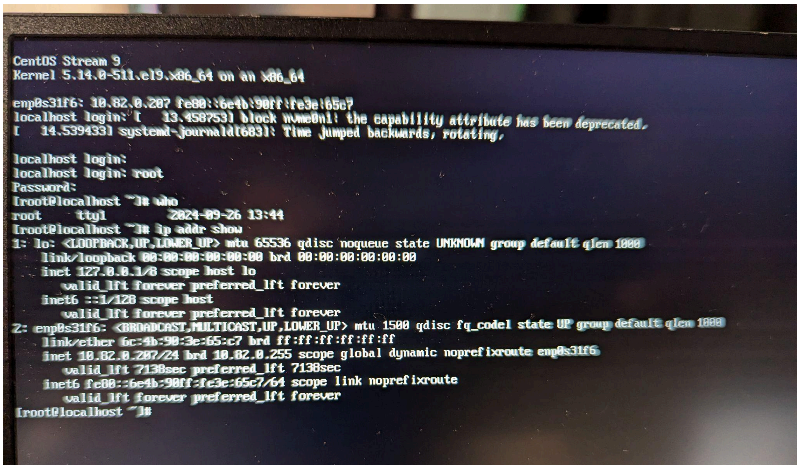
```
CentOS Stream 9
Kernel 5.14.0-511.el9.x86_64 on an x86_64

enp0s31f6: 10.82.0.207 fe80::6e4b:90ff:fe3e:65c7
localhost login: root
Password:
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs            16G     0   16G   0% /dev/shm
tmpfs           6.3G  9.0M  6.3G   1% /run
efivarfs        256K   33K  219K  13% /sys/firmware/efi/efivars
/dev/nvme0n1p3  930G  8.4G  922G   1% /sysroot
composefs       8.4M  8.4M     0 100% /
tmpfs            16G     0   16G   0% /tmp
/dev/nvme0n1p2  960M  133M  828M  14% /boot
/dev/nvme0n1p1  599M  7.5M  592M   2% /boot/efi
tmpfs           3.2G     0  3.2G   0% /run/user/0
[root@localhost ~]#
```

First login to our new appliance

3. Next, let's determine the IP address of our new system.

(command)
$ ip addr show

Your output should look something like this - make note of your IP address.



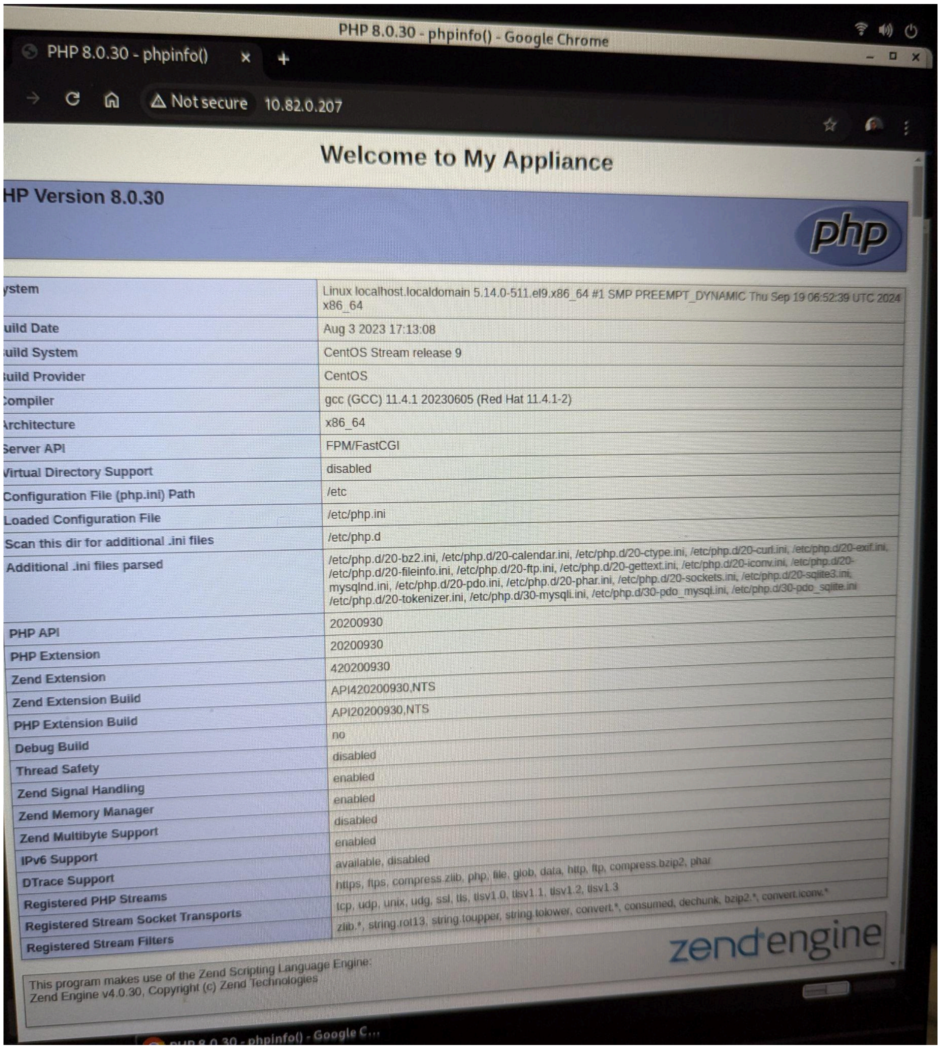
```
CentOS Stream 9
Kernel 5.14.0-511.el9.x86_64 on an x86_64

enp0s31f6: 10.82.0.207 fe80::6e4b:90ff:fe3e:65c7
localhost login: [   13.458753] block nvme0n1: the capability attribute has been deprecated.
[   14.539433] systemd-journald[683]: Time jumped backwards, rotating.

localhost login:
localhost login: root
Password:
[root@localhost ~]# who
root     tty1         2024-09-26 13:44
[root@localhost ~]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s31f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 6c:4b:90:3e:65:c7 brd ff:ff:ff:ff:ff:ff
    inet 10.82.0.207/24 brd 10.82.0.255 scope global dynamic noprefixroute enp0s31f6
       valid_lft 7138sec preferred_lft 7138sec
    inet6 fe80::6e4b:90ff:fe3e:65c7/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@localhost ~]#
```

Getting your appliance's IP Address

4. Now let's open a web browser on another machine and test the LAMP stack.

Go to the IP address that you found in the previous step.

Your result should look like this.


PHP 8.0.30 - phpinfo() - Google Chrome
PHP 8.0.30 - phpinfo()
Not secure
10.82.0.207
php
zend engine
PHP 8.0.30 - phpinfo() - Google C...

Welcome to My Appliance

HP Version 8.0.30

ystem	Linux localhost.localdomain 5.14.0-511.el9.x86_64 #1 SMP PREEMPT_DYNAMIC Thu Sep 19 06:52:39 UTC 2024 x86_64
uild Date	Aug 3 2023 17:13:08
uild System	CentOS Stream release 9
uild Provider	CentOS
ompiler	gcc (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)
rchitecture	x86_64
erver API	FPM/FastCGI
irtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-mysqlnd.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/30-mysqli.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, compress.bzip2, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, convert.*, consumed, dechunk, bzip2.*, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v4.0.30, Copyright (c) Zend Technologies

Viewing your appliance's application from another machine

Welcome to the new world where if you can create a container, you can build a whole system. Let's now move onto how we update these awesome beasts.

Exercise 5: Create an Updated Container

In this exercise, we will make updates to our previously built container image, which will in turn provide updates to our Image Mode machine.

1. Your new appliance is defined by its container image. To create up update for your appliance, all we need to do is create a new container (then publish it to our registry). In this step, We will start by creating a new container file called **mycontainerfile2.cf**.
   (command)
   $ vi mycontainerfile2.cf

The contents of your file should look like this. Don't forget to save the file.

```
FROM quay.io/centos-bootc/centos-bootc:stream9

RUN dnf install -y httpd mariadb mariadb-server php-fpm php-mysqlnd && dnf clean all

RUN systemctl enable httpd mariadb php-fpm

#this next command is all one line although looks like two

RUN echo '<h1 style="Text-align:center;">Welcome to My Appliance</h1><?php phpinfo(); ?>' >>
/var/www/html/index.php

# new stuff

RUN dnf install -y cockpit

RUN systemctl enable cockpit.socket
```

2. Build the new version of your container image.
   NOTE: Replace your own Quay.io username where *[my_account]* appears in the command line. You may also have to login to quay before running this command (see exercise 1 - step 13).
   (command)
   $ podman build -f mycontainerfile2.cf -t quay.io/[my_account]/lamp-bootc:latest

   (begin output)

```
        (((output truncated)))

Installed products updated.

Installed:
  PackageKit-1.2.6-1.el9.x86_64
  PackageKit-glib-1.2.6-1.el9.x86_64
  abattis-cantarell-fonts-0.301-4.el9.noarch
  adobe-source-code-pro-fonts-2.030.1.050-12.el9.1.noarch
  audit-3.1.5-1.el9.x86_64
  centos-logos-90.8-1.el9.x86_64
  clevis-20-200.el9.x86_64
  clevis-luks-20-200.el9.x86_64
  cockpit-323.1-1.el9.x86_64
  cockpit-bridge-323.1-1.el9.x86_64
  cockpit-packagekit-323.1-1.el9.noarch
  cockpit-storaged-323.1-1.el9.noarch
  cockpit-system-323.1-1.el9.noarch
  cockpit-ws-323.1-1.el9.x86_64
  device-mapper-multipath-0.8.7-32.el9.x86_64
  device-mapper-multipath-libs-0.8.7-32.el9.x86_64
  fonts-filesystem-1:2.0.5-7.el9.1.noarch
  gdk-pixbuf2-2.42.6-4.el9.x86_64
  glib-networking-2.68.3-3.el9.x86_64
  gsettings-desktop-schemas-40.0-6.el9.x86_64
  initscripts-service-10.11.8-4.el9.noarch
  iscsi-initiator-utils-6.2.1.9-1.gita65a472.el9.x86_64
  iscsi-initiator-utils-iscsiuio-6.2.1.9-1.gita65a472.el9.x86_64
  isns-utils-libs-0.101-4.el9.x86_64
  jose-14-1.el9.x86_64
  libappstream-glib-0.7.18-5.el9.x86_64
  libatomic-11.5.0-2.el9.x86_64
  libblockdev-lvm-2.28-10.el9.x86_64
  libjose-14-1.el9.x86_64
  libjpeg-turbo-2.0.90-7.el9.x86_64
  libluksmeta-9-12.el9.x86_64
  libpng-2:1.6.37-12.el9.x86_64
  libproxy-0.4.15-35.el9.x86_64
  libproxy-webkitgtk4-0.4.15-35.el9.x86_64
  libsoup-2.72.0-8.el9.x86_64
  libstemmer-0-18.585svn.el9.x86_64
  luksmeta-9-12.el9.x86_64
  python3-dasbus-1.4-5.el9.noarch
  python3-libxml2-2.9.13-6.el9.x86_64
  python3-psutil-5.8.0-12.el9.x86_64
  python3-tracer-1.1-2.el9.noarch
  setroubleshoot-plugins-3.3.14-4.el9.noarch
  setroubleshoot-server-3.3.32-1.el9.x86_64
  sscg-3.0.0-7.el9.x86_64
  tracer-common-1.1-2.el9.noarch
  udisks2-iscsi-2.9.4-11.el9.x86_64
  udisks2-lvm2-2.9.4-11.el9.x86_64
  webkit2gtk3-jsc-2.44.3-2.el9.x86_64

Complete!
```

```
--> 6ab95e317a3c
STEP 6/6: RUN systemctl enable cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/cockpit.socket →
/usr/lib/systemd/system/cockpit.socket.
COMMIT quay.io/matt_st_onge/bootc/lamp-bootc:latest
--> fe247cf7e89d
Successfully tagged quay.io/matt_st_onge/bootc/lamp-bootc:latest
fe247cf7e89d97d5832d889718750d63cc5f2f24dcfd5ed4cce39dfafd150778
```

(end output)

3. Now that you’ve rebuilt your container image. Feel free to test it if you like the same way we did in a previous exercise or don’t (it’s optional). We do, however, have to push this new image to our registry and set it as the latest version.
   NOTE: Replace your own Quay.io username where *[my_account]* appears in the command line.
   (command)
   ```
   $ podman push quay.io/[my_account]/bootc/lamp-bootc:latest
   ```

   (begin output)

   ```
             (((output truncated)))

   Copying blob 8c789e616763 skipped: already exists
   Copying blob fd730fb4a24b skipped: already exists
   Copying blob 232fb94490b0 skipped: already exists
   Copying blob 54246c915569 skipped: already exists
   Copying blob ad312c5c40cc skipped: already exists
   Copying blob bd9ddc54bea9 skipped: already exists
   Copying blob 386e8ecea514 done   |
   Copying blob 2463de35bc3e skipped: already exists
   Copying blob d4cfe3c3d422 skipped: already exists
   Copying blob eedcea4f81f6 done   |
   Copying blob 2bca4ceb08f4 skipped: already exists
   Copying config fe247cf7e8 done   |
   Writing manifest to image destination
   ```

   (end output)

Wow! This is all that you have to do if you want your system to pick up an update automatically. As we are impatient creatures, let’s move onto the next exercise and force the update manually.

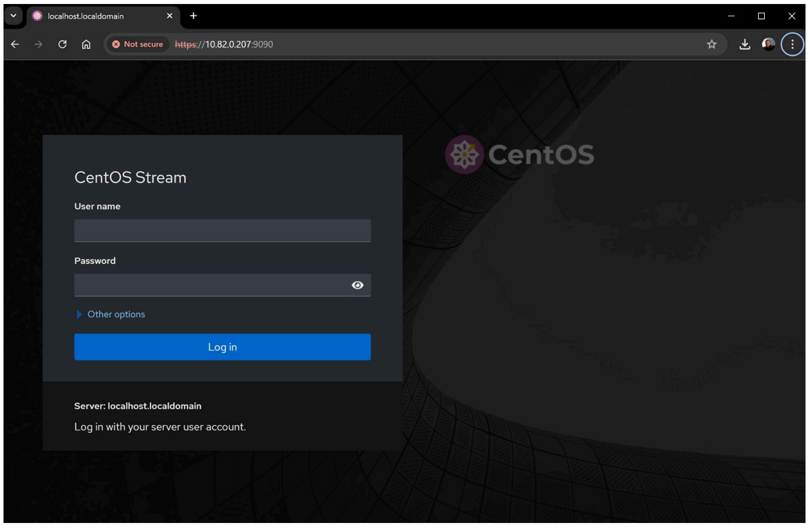
Exercise 6: Update Your System

In this exercise, we will leverage the latest updates to the container you have created to improve and update our bootable container (boot_c) machine.
NOTE: your machine will check for updates automatically every few hours. The default time check period can be modified.

1. Log back into the console of your appliance machine. Run the following as root.
   (command)
   ```
   #  /usr/bin/bootc update –apply –quiet
   ```

Your machine will pull down its updates and reboot itself automatically.
Well done! You have not only created your first boot_c machine, but you have established an update mechanism and successfully updated your new machine. Congratulations.

You added the web console to your image based appliance. Although you cannot log in as root, I hope you know that you can add additional users in the kickstart if you want to rebuild or you can add a user in your console now. Here we only wanted to show just how easy it is to create an update. It works… Gorgeously.



Server: localhost.localdomain
Log in with your server user account.

Appliance is updated with new functionality

I hope that you have enjoyed walking through these exercises and that they have inspired thought as to how you could leverage this technology to build a better appliance.